MCAL User Manual for Uart

32-bit TriCore™ AURIX™ TC3xx microcontroller

About this document

Scope and purpose

This User Manual is intended to enable users to integrate the Microcontroller Abstraction Layer (MCAL) software for the TriCore™ AURIX™ family of 32-bit microcontrollers.

This document describes responsibilities of integrator in-charge of integrating MCAL software with the basic software (BSW) stack. This document also provides detailed information on safety, configuration and functions along with examples of usage of significant features.

***Note**: Detailed information about package installation, safety and other generic information that are common across all modules are provided in MCAL User Manual General.*

Intended audience

This document is intended for anyone using the Uart module of the TC3xx MCAL software.

Document conventions

Table 1 Conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, table and figure captions, screen names, windows, dialog boxes, menus, sub-menus
Italics	Denotes variable(s) and reference(s)
`Courier New`	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, file/folder names, directories, command line inputs, code snippets
>	Indicates that a cascading sub-menu opens when you select a menu item
[cover parentID=<alpha numeric value>]	Used for traceability completeness. Reader should ignore these.

Reference documents

This User Manual should be read in conjunction with the following documents:

- AURIX™ TC3xx MCAL User Manual General

Table of contents

About this document . . . 1
Table of contents . . . 2
1 Uart driver . . . 6
1.1 User information . . . 6
1.1.1 Description . . . 6
1.1.2 Hardware-software mapping . . . 6
1.1.2.1 SRC: dependent hardware peripheral . . . 6
1.1.2.2 PORT: dependent hardware peripheral . . . 7
1.1.2.3 SCU: dependent hardware peripheral . . . 7
1.1.2.4 ASCLIN: primary hardware peripheral . . . 7
1.1.3 File structure . . . 8
1.1.3.1 C file structure . . . 8
1.1.3.2 Code generator plugin files . . . 10
1.1.4 Integration hints . . . 11
1.1.4.1 Integration with AUTOSAR stack . . . 11
1.1.4.2 Multicore and Resource Manager . . . 14
1.1.4.3 MCU support . . . 14
1.1.4.4 Port support . . . 15
1.1.4.5 DMA support . . . 17
1.1.4.6 Interrupt connections . . . 17
1.1.4.7 Example usage . . . 21
1.1.5 Key architectural considerations . . . 41
1.2 Assumptions of Use (AoU) . . . 42
1.3 Reference information . . . 44
1.3.1 Configuration interfaces . . . 44
1.3.1.1 Container: CommonPublishedInformation . . . 44
1.3.1.1.1 ArMajorVersion . . . 44
1.3.1.1.2 ArMinorVersion . . . 45
1.3.1.1.3 ArPatchVersion . . . 45
1.3.1.1.4 ModuleId . . . 46
1.3.1.1.5 Release . . . 46
1.3.1.1.6 SWMajorVersion . . . 46
1.3.1.1.7 SWMinorVersion . . . 47
1.3.1.1.8 SWPatchVersion . . . 47
1.3.1.1.9 VendorId . . . 48
1.3.1.2 Container: UartChannel . . . 48
1.3.1.2.1 UartAutoCalcBaudParams . . . 48
1.3.1.2.2 UartBaudRate . . . 49
1.3.1.2.3 UartCTSEnable . . . 50

Table of contents

1.3.1.2.4 UartCTSPinSelection 50
1.3.1.2.5 UartCTSPolarity 51
1.3.1.2.6 UartChanBaudDenominator 51
1.3.1.2.7 UartChanBaudNumerator 52
1.3.1.2.8 UartChanBaudOverSampling 53
1.3.1.2.9 UartChanBaudPrescalar 54
1.3.1.2.10 UartChannelId 54
1.3.1.2.11 UartDataLength 55
1.3.1.2.12 UartFrameSpace 55
1.3.1.2.13 UartHwUnit 56
1.3.1.2.14 UartParityBit 56
1.3.1.2.15 UartRxChannelMode 57
1.3.1.2.16 UartRxPinSelection 57
1.3.1.2.17 UartStopBits 58
1.3.1.2.18 UartTxChannelMode 59
1.3.1.3 Container: UartConfigSet 59
1.3.1.4 Container: UartGeneral 59
1.3.1.4.1 UartAbortReadApi 59
1.3.1.4.2 UartAbortWriteApi 60
1.3.1.4.3 UartClockRef 60
1.3.1.4.4 UartCsrClksel 61
1.3.1.4.5 UartDeInitApi 62
1.3.1.4.6 UartDevErrorDetect 62
1.3.1.4.7 UartIndex 63
1.3.1.4.8 UartInitCheckApi 63
1.3.1.4.9 UartInitDeInitApiMode 64
1.3.1.4.10 UartMainFunctionReadPeriod 64
1.3.1.4.11 UartMainFunctionWritePeriod 65
1.3.1.4.12 UartRunTimeErrorDetect 65
1.3.1.4.13 UartSafetyEnable 66
1.3.1.4.14 UartSleepEnable 66
1.3.1.4.15 UartStreamingRecvModeApi 67
1.3.1.4.16 UartTimeoutCount 67
1.3.1.4.17 UartVersionInfoApi 68
1.3.1.5 Container: UartNotification 68
1.3.1.5.1 UartAbortReceiveNotifPtr 69
1.3.1.5.2 UartAbortTransmitNotifPtr 69
1.3.1.5.3 UartReceiveNotifPtr 70
1.3.1.5.4 UartStreamingRecvNotifPtr 70
1.3.1.5.5 UartTransmitNotifPtr 71
1.3.1.6 Container: Uart 71
1.3.2 Functions - Type definitions 72

Table of contents

1.3.2.1 Uart_ChannelIdType . . . 72
1.3.2.2 Uart_ConfigType . . . 72
1.3.2.3 Uart_ErrorIdType . . . 72
1.3.2.4 Uart_MemType . . . 73
1.3.2.5 Uart_NotificationPtrType . . . 73
1.3.2.6 Uart_ReturnType . . . 73
1.3.2.7 Uart_SizeType . . . 74
1.3.2.8 Uart_StatusType . . . 74
1.3.2.9 Uart_StreamingRecvNotiPtrType . . . 74
1.3.2.10 UartNotificationCallback . . . 75
1.3.2.11 UartStreamingRecvNotifPtr . . . 75
1.3.3 Functions - APIs . . . 75
1.3.3.1 Uart_InitCheck . . . 75
1.3.3.2 Uart_StartStreaming . . . 76
1.3.3.3 Uart_StopStreaming . . . 78
1.3.3.4 Uart_Init . . . 79
1.3.3.5 Uart_Read . . . 79
1.3.3.6 Uart_Write . . . 81
1.3.3.7 Uart_AbortRead . . . 82
1.3.3.8 Uart_AbortWrite . . . 83
1.3.3.9 Uart_GetStatus . . . 83
1.3.3.10 Uart_DeInit . . . 84
1.3.3.11 Uart_GetVersionInfo . . . 85
1.3.4 Notifications and Callbacks . . . 86
1.3.5 Scheduled functions . . . 86
1.3.5.1 Uart_MainFunction_Read . . . 86
1.3.5.2 Uart_MainFunction_Write . . . 87
1.3.6 Interrupt service routines . . . 88
1.3.6.1 Uart_IsrError . . . 88
1.3.6.2 Uart_IsrReceive . . . 89
1.3.6.3 Uart_IsrTransmit . . . 90
1.3.7 Callout . . . 90
1.3.8 Errors Handling . . . 91
1.3.9 Deviations and limitations . . . 92
1.3.9.1 Deviations . . . 92
1.3.9.1.1 Software specification deviations . . . 92
1.3.9.1.2 AMDC Violations . . . 92
1.3.9.1.3 VSMD Violations . . . 92
1.3.9.2 Limitations . . . 92
Revision history . . . 95

Table of contents

Disclaimer 97

1 Uart driver

1.1 User information

1.1.1 Description

The UART driver is responsible for providing communication services as per the UART protocol. The ASCLIN module provides hardware support for asynchronous communication to realize the UART protocol. The UART driver provides functionality for configuration, initialization, data transmission, reception and also provides optional features such as abort transmission and abort reception.

1.1.2 Hardware-software mapping

This section describes the system view of the UART driver and peripherals administered by it.

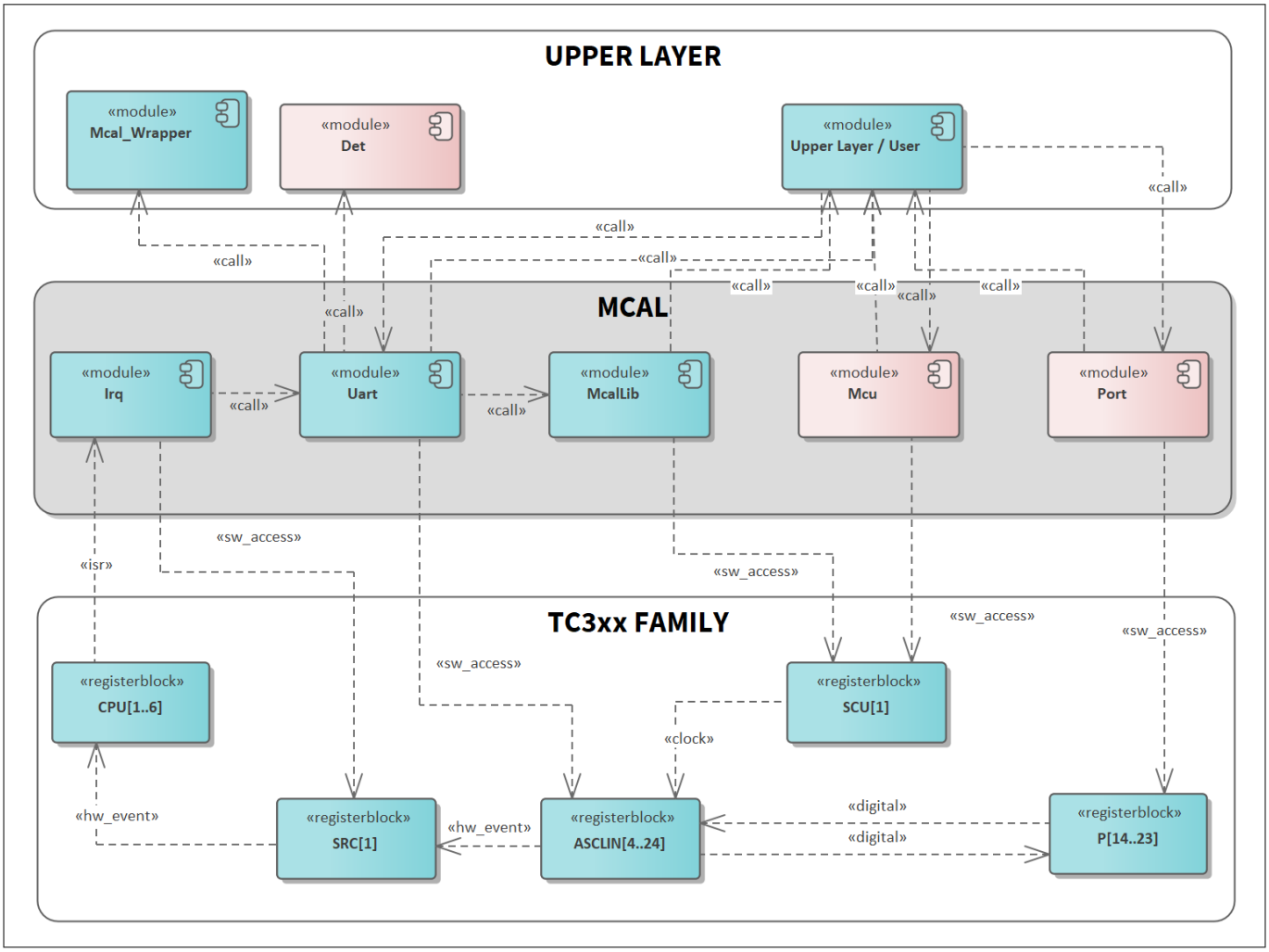


Figure 1 Mapping of hardware-software interfaces

1.1.2.1 SRC: dependent hardware peripheral

Hardware functional features

The UART driver depends on the interrupt router for raising an interrupt to the CPU based on the transmit and receive events, which indicates successful data transmission and reception respectively.

Users of the hardware

The interrupt router is configured either by the IRQ driver or the user software. No functional block of the interrupt router is administered by the UART driver.

Hardware diagnostic features

The SMU alarms configured for interrupt router are not monitored by the UART driver.

Hardware events

The interrupt events raised by the interrupt router are serviced by the CPU. The UART driver provides interrupt handlers as software interfaces, which must be invoked from the ISR.

1.1.2.2 PORT: dependent hardware peripheral

Hardware functional features

The ARX, ATX, CTS and RTS signals are routed to the ASCLIN through the port pads. These signals are configured and enabled through the PORT driver.

Users of the hardware

The port pads are configured by the PORT driver.

Hardware diagnostic features

Not applicable.

Hardware events

Hardware events from port pads are not used by the UART driver.

1.1.2.3 SCU: dependent hardware peripheral

Hardware functional features

The UART driver depends on the SCU IP for the clock, ENDINIT and reset functionalities. The driver requires the fSPB, fASCLINF and fASCLINS clock signals for functioning.

Users of the hardware

The SCU IP supplies clock for all the peripherals and the MCU driver is responsible for configuring the clock tree. To avoid conflicts due to simultaneous writes, update to all the ENDINIT protected registers is performed using the MCALLIB APIs.

Hardware diagnostic features

The SMU alarms configured for the SCU IP are not monitored by the UART driver.

Hardware events

Hardware events from the SCU are not used by the UART driver.

1.1.2.4 ASCLIN: primary hardware peripheral

Hardware functional features

The UART driver uses the ASCLIN for transmission and reception of data. The key hardware functional features used by the driver are:

- Full-duplex asynchronous operating modes
- Supports half duplex operating mode

- 16 bytes TXFIFO
- 16 bytes RXFIFO
- 2 to 16 bits data frames
- Parity-bit generation/checking
- One or two stop bits
- Baud rate configuration
- Optional RTS / CTS handshaking
- Programmable over-sampling 4 to 16 times per bit
- Programmable sample point position
- Interrupt generation
- Interrupt signals capable of triggering a CPU
- Programmable digital glitch filter and median filter for incoming bit stream
- Pack / unpack capabilities of the Tx and Rx FIFOs
- Shift direction LSB first for ASC

The unsupported feature of the ASCLIN is:

- Internal loop-back mode

Users of the hardware

The LIN and UART drivers utilizes the ASCLIN IP. The allocation of ASLIN channels to LIN/UART driver is done by the MCU driver. Both LIN and UART drivers utilize only the channels allocated to them.

Hardware diagnostic features

The SMU alarms configured for the ASCLIN are not monitored by the UART driver.

Hardware events

The UART driver uses the following hardware events from the ASCLIN IP:

- Transmit FIFO level interrupt: This interrupt is triggered when TXFIFO transmission is completed successfully.
- Receive FIFO level interrupt: This interrupt is triggered when RXFIFO filled up to configured level with received data.
- Error condition parity error, frame error, RXFIFO overflow error.

1.1.3 File structure

1.1.3.1 C file structure

This section provides details of the C files of the UART driver.

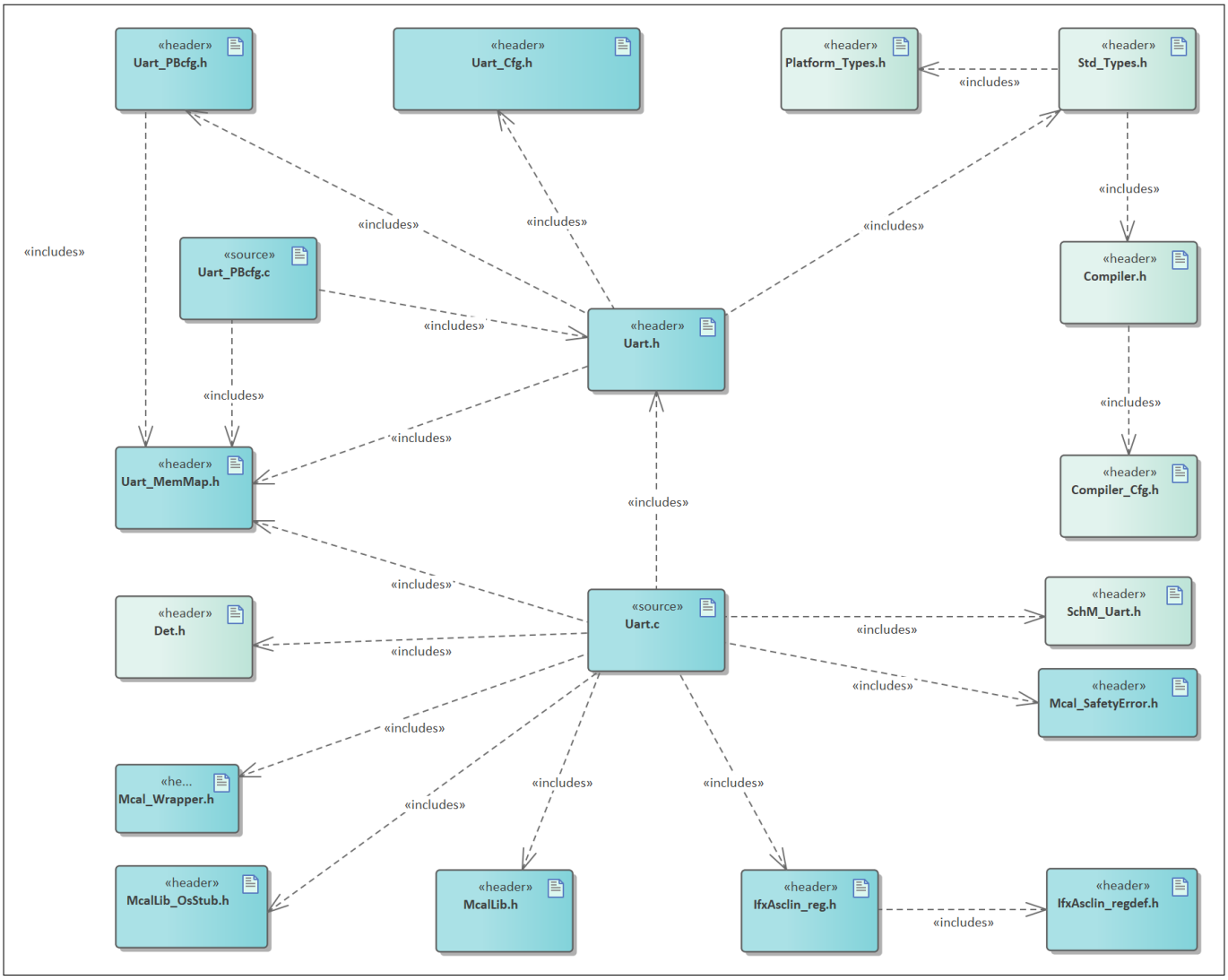


Figure 2 Uart_C_File_Structure-1.png

Table 2 C file structure

File name	Description
`Compiler.h`	Provides abstraction from compiler-specific keywords
`Compiler_Cfg.h`	Configuration header file for compiler abstraction
`Det.h`	Provides the exported interfaces of Development Error Tracer
`IfxAsclin_reg.h`	SFR header file for ASCLIN
`IfxAsclin_regdef.h`	SFR header file for ASCLIN
`McalLib.h`	Static header file defining prototypes of data structure and APIs exported by the MCALLIB.
`McalLib_OsStub.h`	McalLib_OsStub.h provides macros to support user mode of Tricore. This shall be included by other drivers to call OS APIs.
`Mcal_SafetyError.h`	Header file containing the prototype of the API for reporting safety-related errors

(table continues...)

Table 2 (continued) C file structure

File name	Description
Mcal_Wrapper.h	Provides the exported interfaces for Production Error and Runtime Development Errors. Implemented by default to include functions of Dem.h and Det.h files. This file can be modified by the user but function prototype is not user modifiable.
Platform_Types.h	Platform-specific type declaration file as defined by AUTOSAR
SchM_Uart.h	Header file containing prototype of the scheduled function of the Uart driver.
Std_Types.h	Standard type declaration file as defined by AUTOSAR. It is independent of compiler or platform.
Uart.c	File (static) containing implementation of APIs and ISRs
Uart.h	Header file (Static) defining prototypes of data structures, APIs and Interrupt handlers
Uart_Cfg.h	Contains UART driver pre-compile configuration parameters
Uart_MemMap.h	File containing the memory section definitions used by the UART driver
Uart_PBcfg.c	File (generated) containing objects to data structures
Uart_PBcfg.h	Post-build header file for the UART driver

1.1.3.2 Code generator plugin files

This section provides details of the code generator plugin files of the UART driver.

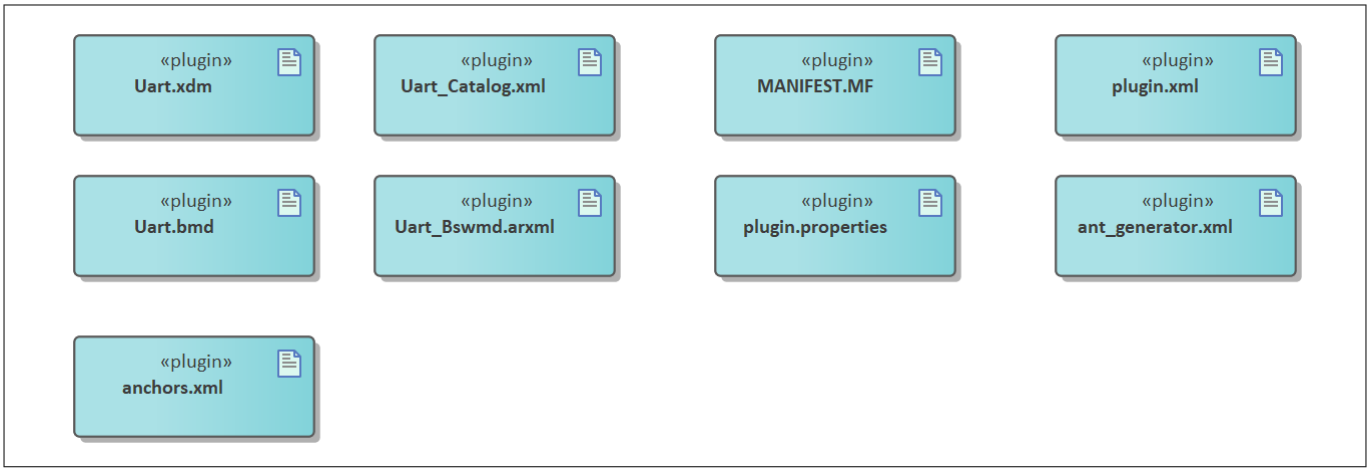


Figure 3 Uart_Code_Generator_Plugin_Files-1.png

Table 3 Code generator plugin files

File name	Description
MANIFEST.MF	Tresos plugin support file containing the metadata for UART driver
Uart.bmd	AUTOSAR format XML data model schema file (for each device)
Uart.xdm	Tresos format XML data model schema file
Uart_Bswmd.arxml	AUTOSAR format module description file
Uart_Catalog.xml	AUTOSAR format catalog file

(table continues...)

Table 3 (continued) Code generator plugin files

File name	Description
`anchors.xml`	Tresos anchors support file for the UART driver
`ant_generator.xml`	Tresos support file to generate and rename multiple post-build configurations when using variation point
`plugin.properties`	Tresos plugin support file for the UART driver
`plugin.xml`	Tresos plugin support file for the UART driver

1.1.4 Integration hints

This section lists the key points that an integrator or user of the UART driver must consider.

1.1.4.1 Integration with AUTOSAR stack

This section lists the modules, which are not part of MCAL, but required to integrate the UART driver.

- **EcuM**

  The ECU Manager module is a part of the AUTOSAR stack that manages common aspects of ECU. Specifically, in the context of MCAL, EcuM is used for initialization and de-initialization of the software drivers. The EcuM module provided in the MCAL package is a stub code and needs to be replaced with a complete EcuM module during the integration phase.

- **Memory mapping**

  Memory mapping is a concept from AUTOSAR that allows relocation of text, variables, constants and configuration data to user-specific memory regions. To achieve this, all the relocatable elements of the driver are encapsulated in different memory-section macros. These macros are defined in the `Uart_MemMap.h` file.

  The `Uart_MemMap.h` file is provided in the MCAL package as a stub code. The integrator must place appropriate compiler pragmas within the memory-section macros. The pragmas ensure that the elements are re-located to the correct memory region. A sample implementation listing the memory-section macros is shown as follows.

```
/*To be used for all global or static variables.*/
#if defined UART_START_SEC_VAR_CLEARED_ASIL_B_LOCAL_8
 /* User Pragma here */
 #undef UART_START_SEC_VAR_CLEARED_ASIL_B_LOCAL_8
 #undef MEMMAP_ERROR
#elif defined UART_STOP_SEC_VAR_CLEARED_ASIL_B_LOCAL_8
 /* User Pragma here */
 #undef UART_STOP_SEC_VAR_CLEARED_ASIL_B_LOCAL_8
 #undef MEMMAP_ERROR
#elif defined UART_START_SEC_VAR_CLEARED_ASIL_B_LOCAL_32
 /* User Pragma here */
 #undef UART_START_SEC_VAR_CLEARED_ASIL_B_LOCAL_32
 #undef MEMMAP_ERROR
#elif defined UART_STOP_SEC_VAR_CLEARED_ASIL_B_LOCAL_32
 #undef UART_STOP_SEC_VAR_CLEARED_ASIL_B_LOCAL_32
 #undef MEMMAP_ERROR
#elif defined UART_START_SEC_VAR_CLEARED_ASIL_B_LOCAL_UNSPECIFIED
 /* User Pragma here */
 #undef UART_START_SEC_VAR_CLEARED_ASIL_B_LOCAL_UNSPECIFIED
 #undef MEMMAP_ERROR
#elif defined UART_STOP_SEC_VAR_CLEARED_ASIL_B_LOCAL_UNSPECIFIED
 /* User Pragma here */
 #undef UART_STOP_SEC_VAR_CLEARED_ASIL_B_LOCAL_UNSPECIFIED
 #undef MEMMAP_ERROR
 /*To be used for global or static constants.*/
#elif defined UART_START_SEC_CONST_ASIL_B_LOCAL_32
 /* User Pragma here */
 #undef UART_START_SEC_CONST_ASIL_B_LOCAL_32
 #undef MEMMAP_ERROR
#elif defined UART_STOP_SEC_CONST_ASIL_B_LOCAL_32
 /* User Pragma here */
 #undef UART_STOP_SEC_CONST_ASIL_B_LOCAL_32
 #undef MEMMAP_ERROR
 /* UART module configuration data */
#elif defined UART_START_SEC_CONFIG_DATA_ASIL_B_LOCAL_UNSPECIFIED
 /* User Pragma here */
 #undef UART_START_SEC_CONFIG_DATA_ASIL_B_LOCAL_UNSPECIFIED
 #undef MEMMAP_ERROR
#elif defined UART_STOP_SEC_CONFIG_DATA_ASIL_B_LOCAL_UNSPECIFIED
 /* User Pragma here */
 #undef UART_STOP_SEC_CONFIG_DATA_ASIL_B_LOCAL_UNSPECIFIED
 #undef MEMMAP_ERROR
 /* Code section */
#elif defined UART_START_SEC_CODE_ASIL_B_LOCAL
 /* User Pragma here */
 #undef UART_START_SEC_CODE_ASIL_B_LOCAL
 #undef MEMMAP_ERROR
#elif defined UART_STOP_SEC_CODE_ASIL_B_LOCAL
 /* User Pragma here */
 #undef UART_STOP_SEC_CODE_ASIL_B_LOCAL
 #undef MEMMAP_ERROR
#endif
```

```
#if defined MEMMAP_ERROR
#error "Uart_MemMap.h, wrong pragma command"
#endif
```

- **DET**

  The DET module is a part of the AUTOSAR stack that handles all the development errors reported by the BSW modules. The UART driver reports all the development errors to the DET module through the `Det_ReportError()`API . The user of the UART driver must process all the errors reported to the DET module through the API `Det_ReportError()`.

  The `Det.h` and `Det.c` files are provided in the MCAL package as a stub code and needs to be replaced with a complete DET module during the integration phase.

- **Mcal_Wrapper**

  This Driver performs reporting of the Production and Runtime errors. The Handling of the reported errors shall be done by the user. The Mcal_Wrapper_Det_ReportRuntimeError() API, Mcal_Wrapper_Dem_SetEventStatus() API and Mcal_Wrapper_Dem_ReportErrorStatus() API are provided in the Mcal_Wrapper.c and Mcal_Wrapper.h files as a stub code, and can be updated by the integrator to handle the reported errors. The files Mcal_Wrapper.c and Mcal_Wrapper.h are user modifiable, Where the function prototype is not user modifiable and by default the Mcal Wrapper function shall calls AUTOSAR DEM and DET Modules.

  The user of the Uart driver shall process Runtime errors reported to the Mcal_Wrapper module. Production errors are not applicable for Uart driver. The interface used for reporting Runtime error in AUTOSAR version 4.4.0 is Mcal_Wrapper_Det_ReportRuntimeError() API. The Mcal_Wrapper.c and Mcal_Wrapper.h files are provided in the MCAL package as a stub code and can be replaced with a user specific Runtime error handling module/s during the integration phase.

- **SchM**

  The SchM is not required for integrating the UART driver.

- **Safety error**

  The UART driver reports all the detected safety errors through the `Mcal_ReportSafetyError()` API.

  The driver performs only detection and reporting of the safety errors. The handling of the reported errors shall be done by the user. The `Mcal_ReportSafetyError()` API is provided in the `Mcal_SafetyError.c` and `Mcal_SafetyError.h` files as a stub code, and must be updated by the integrator to handle the reported errors.

  Note: All DET errors are also reported as safety errors (error code used is same as DET).

- **Notifications and callbacks**

  The UART driver does not implement any notifications. However, the UART driver reports the completion of a transmit, receive, abort transmit and abort receive operation through notification functions.

  If UART driver operates in streaming mode, streaming notification is provided to the user after copying the received data from FIFO to application buffer.

  These notification functions can be configured by the user in EB tresos for each UART channel separately.

- **Operating system(OS)**

  The OS or application must ensure correct type of service and interrupt priority is configured in the SR register. Enabling and disabling of interrupts must also be managed by the OS or application.

  The OS files provided by MCAL package are only an example code and must be updated by the integrator with the actual OS files for the desired function.

1.1.4.2 Multicore and Resource Manager

The UART driver does not support execution on multiple cores simultaneously.

1.1.4.3 MCU support

The UART driver is dependent on the MCU driver for clock configuration and channel allocation services. The initialization of the UART driver must be started only after completing the MCU initialization. The following must be considered while configuring the MCU driver in tresos:

- The fASCLINF or fASCLINS defines the clock frequency for the ASCLIN kernel. To configure clock frequency for fASCLINF or fASCLINS refer to the McuAscLinFastFrequency and McuAscLinSlowFrequency parameters from the MCU driver configuration as follows:

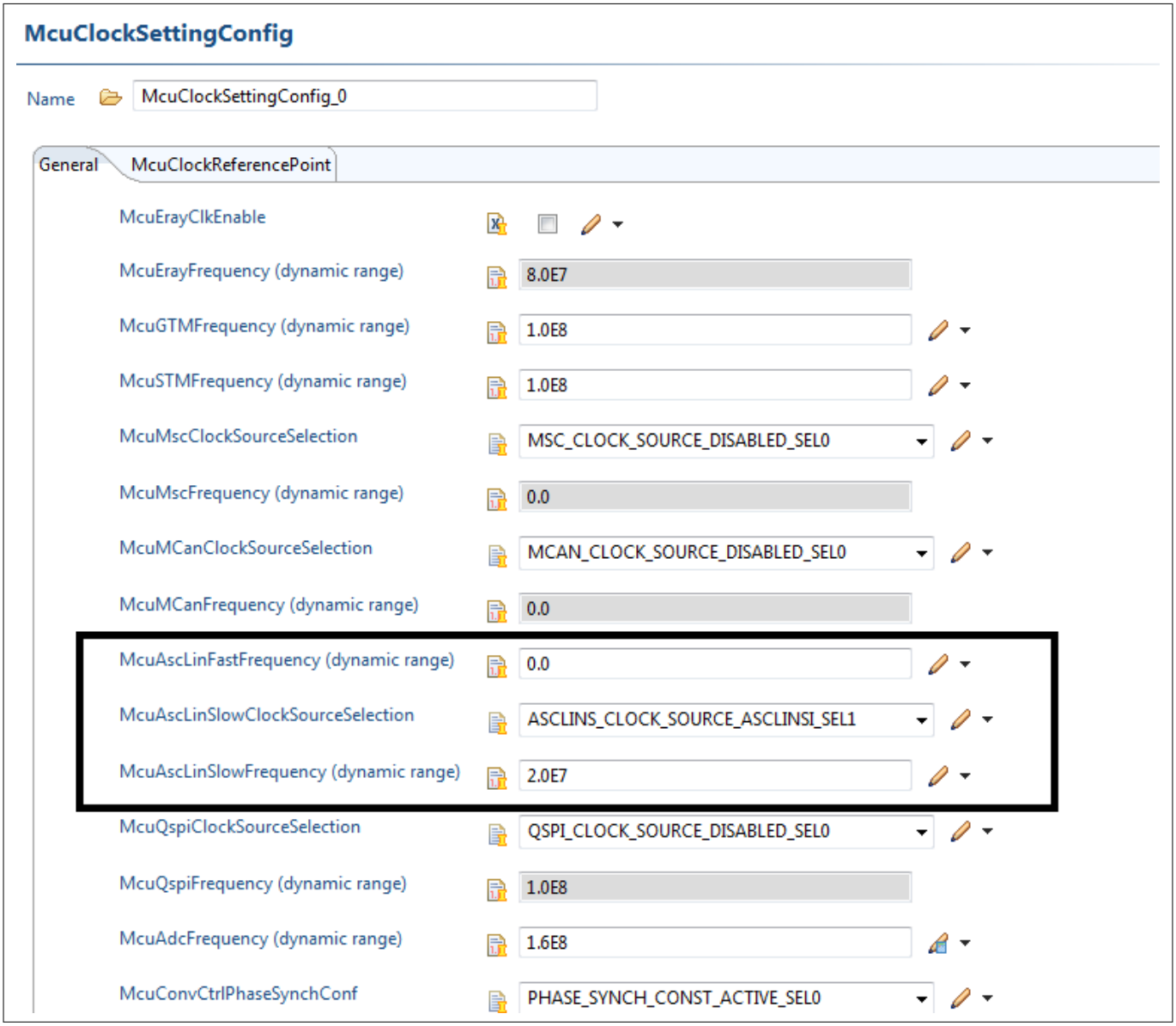
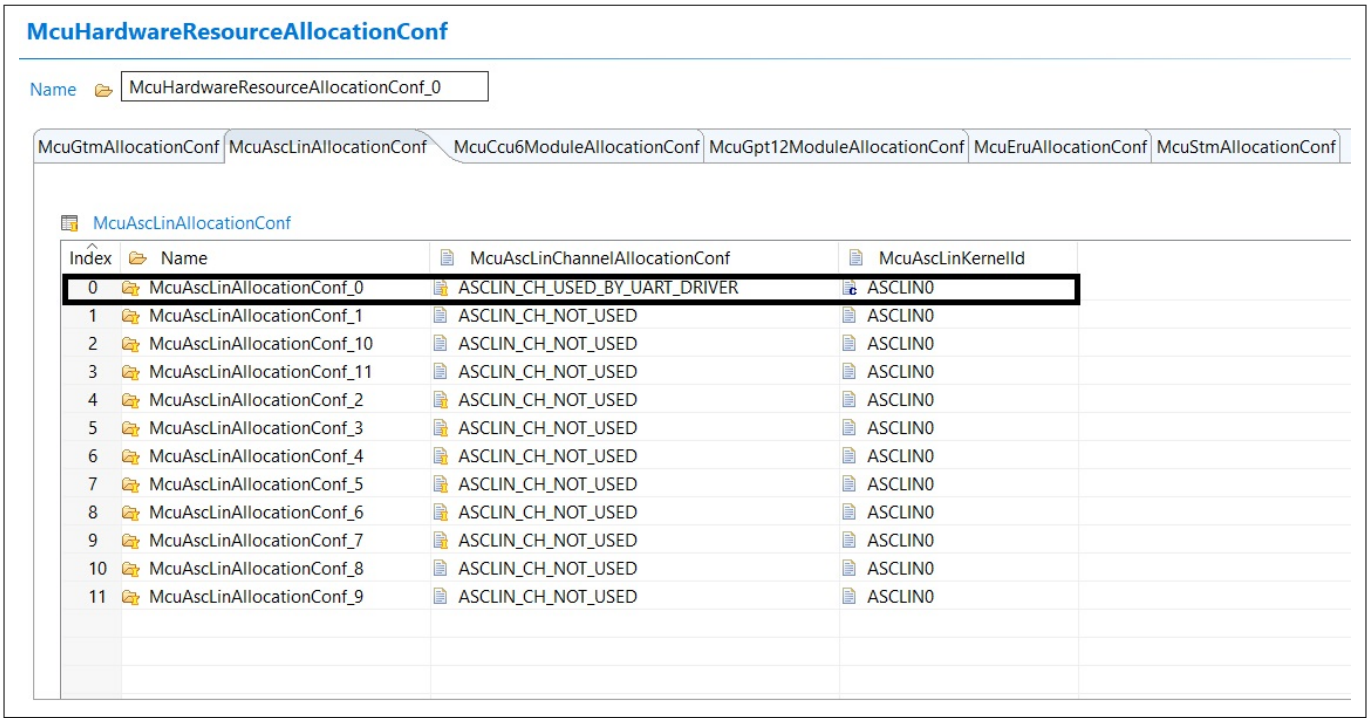


Figure 4 UART fast or slow clock configuration

- The ASCLIN hardware IP is shared between UART and LIN drivers. The resource allocation is configured in the MCU driver. Following is the example of allocation of the ASCLIN0 hardware resource to the UART driver.



Index	Name	McuAscLinChannelAllocationConf	McuAscLinKernelId
0	McuAscLinAllocationConf_0	ASCLIN_CH_USED_BY_UART_DRIVER	ASCLIN0
1	McuAscLinAllocationConf_1	ASCLIN_CH_NOT_USED	ASCLIN0
2	McuAscLinAllocationConf_10	ASCLIN_CH_NOT_USED	ASCLIN0
3	McuAscLinAllocationConf_11	ASCLIN_CH_NOT_USED	ASCLIN0
4	McuAscLinAllocationConf_2	ASCLIN_CH_NOT_USED	ASCLIN0
5	McuAscLinAllocationConf_3	ASCLIN_CH_NOT_USED	ASCLIN0
6	McuAscLinAllocationConf_4	ASCLIN_CH_NOT_USED	ASCLIN0
7	McuAscLinAllocationConf_5	ASCLIN_CH_NOT_USED	ASCLIN0
8	McuAscLinAllocationConf_6	ASCLIN_CH_NOT_USED	ASCLIN0
9	McuAscLinAllocationConf_7	ASCLIN_CH_NOT_USED	ASCLIN0
10	McuAscLinAllocationConf_8	ASCLIN_CH_NOT_USED	ASCLIN0
11	McuAscLinAllocationConf_9	ASCLIN_CH_NOT_USED	ASCLIN0

Figure 5 UART channel allocation in MCU driver

1.1.4.4 Port support

The PORT driver configures the port pins of the entire microcontroller. The user must configure port pins used by the UART driver through the PORT configuration and initialize the port pins prior to invoking of the UART initialization. The following must be considered while configuring PORT driver in the EB tresos tool:

- Configure all port pins that are used in the UART driver for RX, TX, CTS and RTS. That is, parameters such as `PortPinDirection` (input or output), `PortPinInitialMode` (as GPIO for input pin or corresponding ALT option for output pins) and so on.

  Refer to the following sample configurations for the PORT driver:

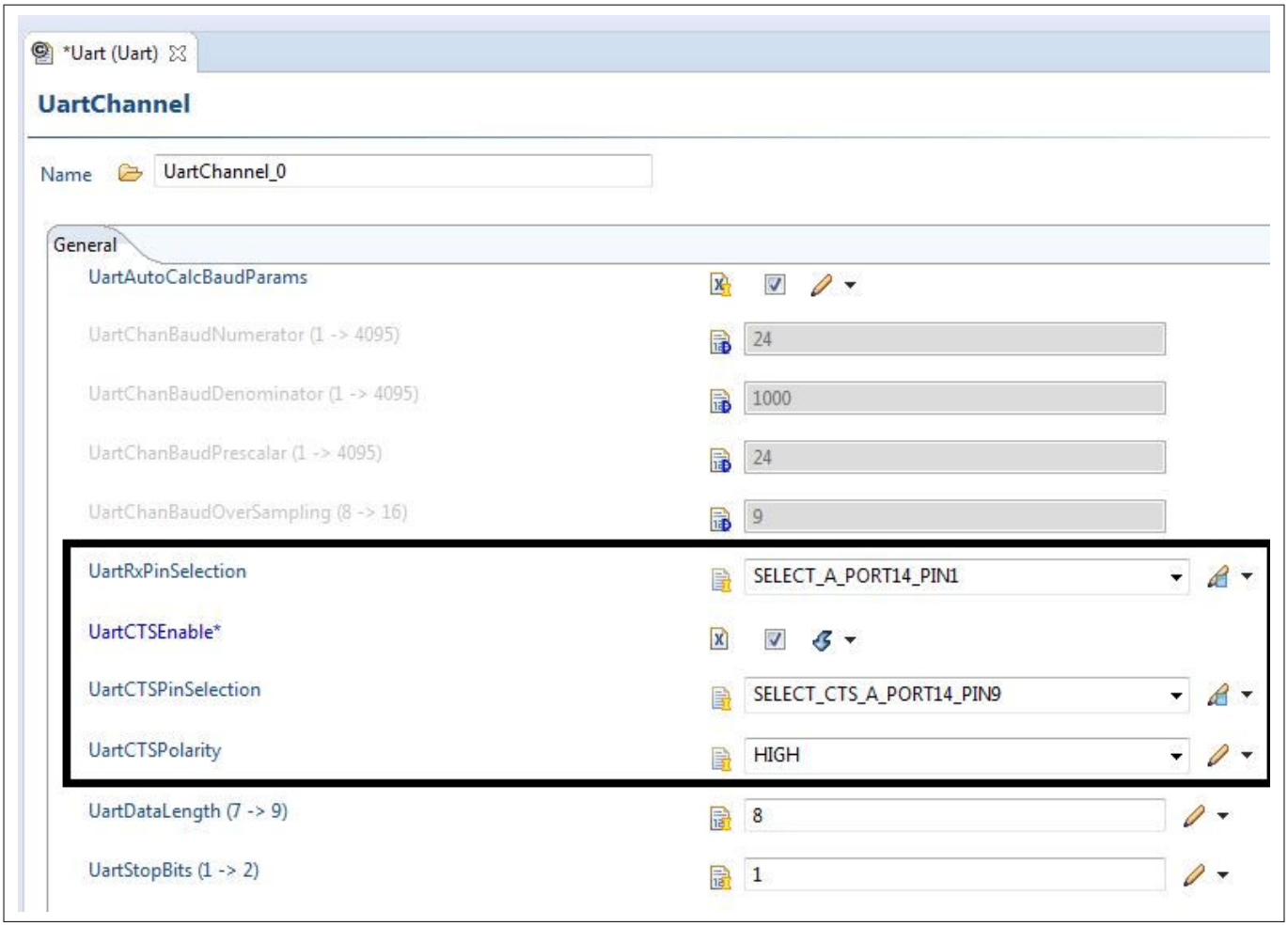


Figure 6 Alternative RX and CTS pin selection for UART channel

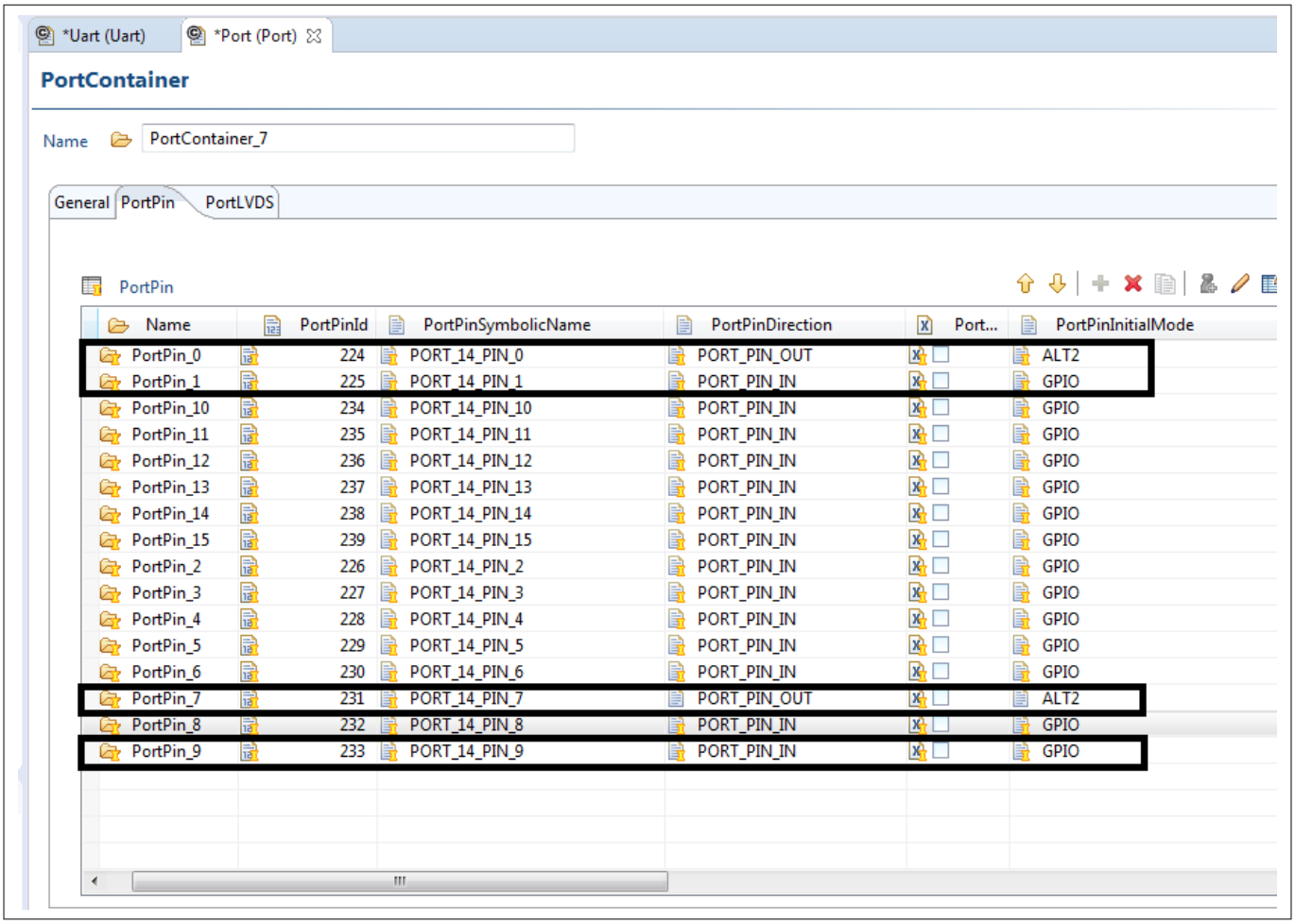


Figure 7 Port pin direction and ALT configuration for UART

1.1.4.5 DMA support

The UART driver does not use any services provided by the DMA driver.

1.1.4.6 Interrupt connections

The interrupt connections of the UART driver are described in this section.

- TFL: TXFIFO level interrupt

This interrupt is triggered when TXFIFO transmission is completed successfully. A sample invocation for TFL interrupt handler is shown as follows:

```
#include "Uart.h"
#include "Irq.h"

/********TX Interrupt for ASCLIN0 *********/

IFX_INTERRUPT(ASCLIN0TX_ISR, 0, IRQ_ASCLIN0_TX_PRIO)
ISR(ASCLIN0TX_ISR)
{
 /* Enable Global Interrupts */
 ENABLE();
 /* Call Uart Interrupt function*/
 Uart_IsrTransmit(0U);
}
```

- RFL: RXFIFO level interrupt

This interrupt is triggered when RXFIFO filled up to configured level with received data.

In normal read operation, `Uart_Read()`API sets the RXFIFO interrupt level based on the received data size.

In streaming mode, `Uart_StartStreaming()`API sets the RXFIFO interrupt level based on the frame size. If configured frame size is 8 bits then RXFIFO interrupt level set to 0 then this interrupt is triggered for each received data byte. If configured frame size is above 8 bits then RXFIFO interrupt level set to 1 then interrupt is triggered for every two received data bytes.

A sample invocation for RFL interrupt handler is shown as follows:

```
#include "Uart.h"
#include "Irq.h"

/********RX Interrupt for ASCLIN0 *********/
IFX_INTERRUPT(ASCLIN0RX_ISR, 0, IRQ_ASCLIN0_RX_PRIO)
ISR(ASCLIN0RX_ISR)
{
 /* Enable Global Interrupts */
 ENABLE();
 /* Call Uart Interrupt function*/
 Uart_IsrReceive(0U);
}
```

- ERR/Transmit complete: Receive error or transmit complete

This interrupt is triggered when receive error (Parity, Frame, RXFIFO overflow) occurred in streaming mode or read mode.

This interrupt is also triggered when transmit complete (last stop bit transmitted out from ASCLIN kernel) event occurs.

A sample invocation for error interrupt handler is shown as follows:

```
#include "Uart.h"
#include "Irq.h"

/********Err Interrupt for ASCLIN0 *********/
IFX_INTERRUPT(ASCLIN0ERR_ISR, 0, IRQ_ASCLIN0_ERR_PRIO)
ISR(ASCLIN0ERR_ISR)
{
 /* Enable Global Interrupts */
 ENABLE();
 /* Call Uart Interrupt function*/
 Uart_IsrError(0U);
}
```

Configuration of interrupt category and priority shall be configured in the IRQ driver. The following examples show interrupt configurations for ASCLIN0:

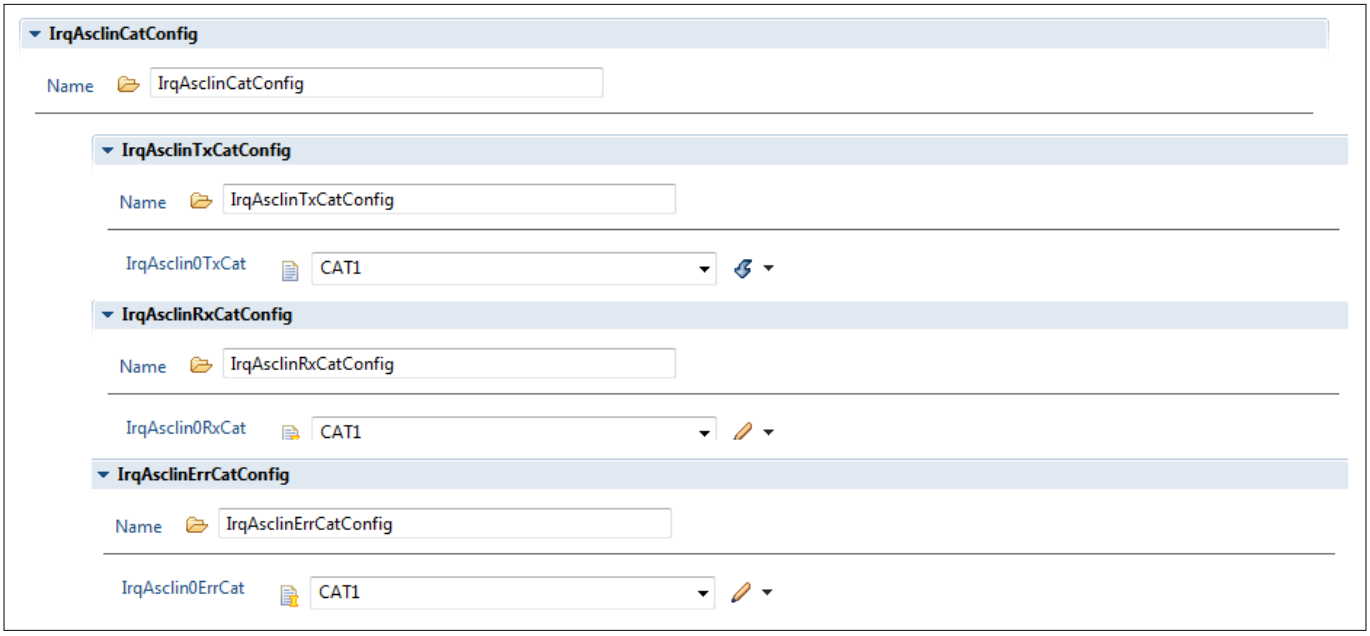


Figure 8 Interrupt category configuration for ASCLIN0

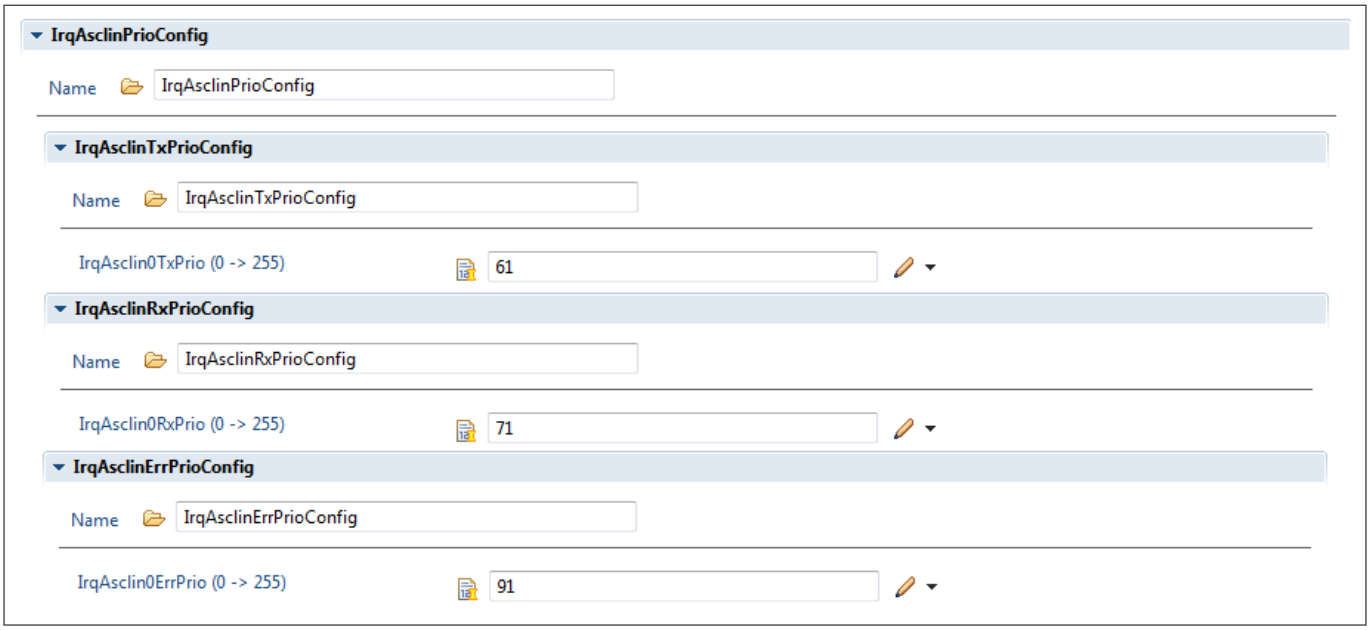


Figure 9 Interrupt priority configuration for ASCLIN0

1.1.4.7 Example usage

The following are some of the key use cases of the UART driver.

Configuration of UART and other modules

- Configuration of system clock: Before using the UART driver, the MCU driver should be configured and initialized so that the system clock is up and running at the required frequency. This configuration is done using the MCU driver.
- Configuration of the port pins: The TXD, RXD, optional CTS, RTS (for the relevant RX and CTS pin select) pins of the UART channels should be configured using the PORT driver.
- Configuration of the UART interrupts: For UART drivers with interrupt mode enabled, configure the interrupt priority, type of service and interrupt type in the IRQ driver.
- Configuration of the UART driver: Select the required API configuration and choose channel dependent parameters like Baud-rate, Tx and Rx mode (polling or interrupt) Stop bits, Parity, Optional CTS selection etc.

UART driver initialization

Refer to the Integration hints section and add all the dependent modules. Follow the sequence in the application code:

1. Initialize the MCU driver and the clock using the `Mcu_Init` API.
2. Initialize the PORT driver using the `Port_Init` API.
3. Initialize the IRQ to enable the interrupt generation.
4. Initialize the UART driver using the `Uart_Init` API.

The sample code for the UART driver initialization is shown as follows:

```
#include "Mcu.h"
#include "Uart.h"
#include "Port.h"
#include "Irq.h"

/* MCU Initialization */
Mcu_Init(&Mcu_Config);
Mcu_InitClock(0U);
while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
Mcu_DistributePllClock();

/* Port initialization */
Port_Init(&Port_Config);

/* Irq initialization */
IrqAsclin_Init();

/* Uart driver initialization */
Uart_Init(&Uart_Config);

/* Check Uart initialization */
RetVal = Uart_InitCheck(&Uart_Config);
if(RetVal == E_NOT_OK)
{
 /* Uart initialization fail */
}

/* Uart driver de-initialization */
Uart_DeInit();
```

UART transmit operation in interrupt mode

The sequence diagram for the UART transmit operation in the interrupt mode is shown as follows:

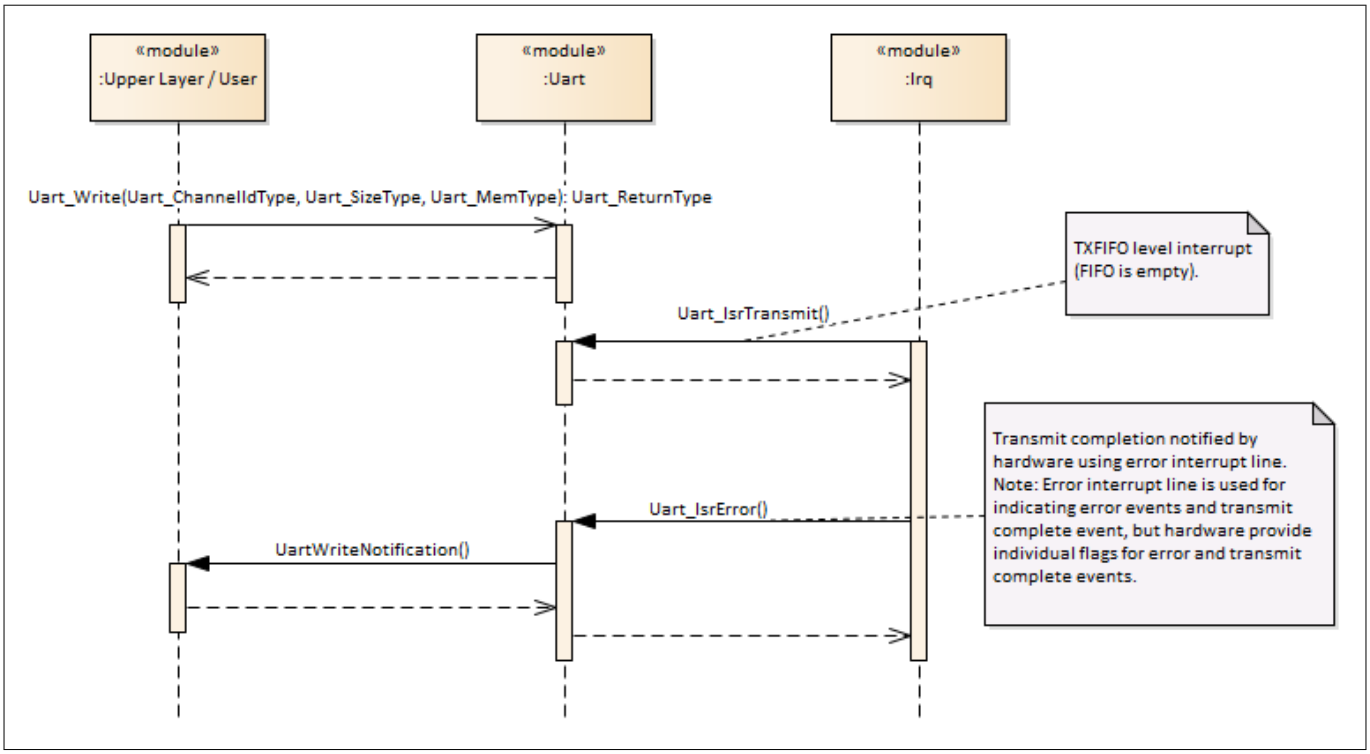


Figure 10 UART transmit operation in interrupt mode

The sample configuration for the UART transmit in the interrupt mode with 8-bit frame length is shown as follows:

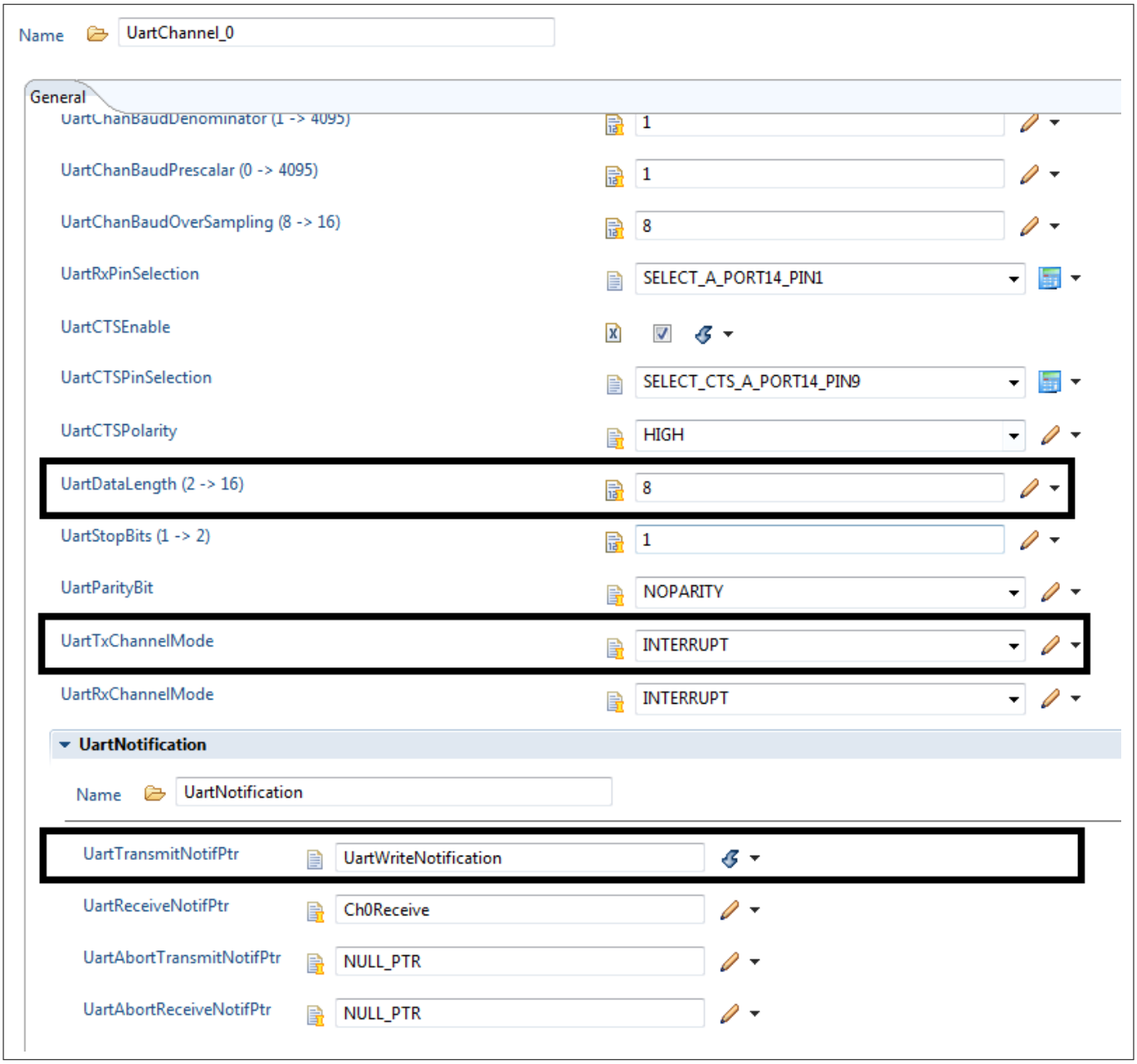


Figure 11 **Configuration: Frame length 8 bits, transmit in interrupt mode, transmit Notification function `UartWriteNotification`**

A sample code for transmitting 20 frames with 8-bit frame length in the interrupt mode is as follows:

```
/* Transmit buffer */
uint8 TxBuffer[20] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};

/* Write notification function */
void UartWriteNotification(Uart_ErrorIdType ErrorId)
{
 if(ErrorId == UART_NO_ERR)
 {
 /* 20 frames transmited successfully */
 }
}

/* Uart write */
Uart_Write(0,&TxBuffer[0],20);
```

UART transmit operation in polling mode

The sequence diagram for the UART transmit operation in the polling mode is shown as follows:

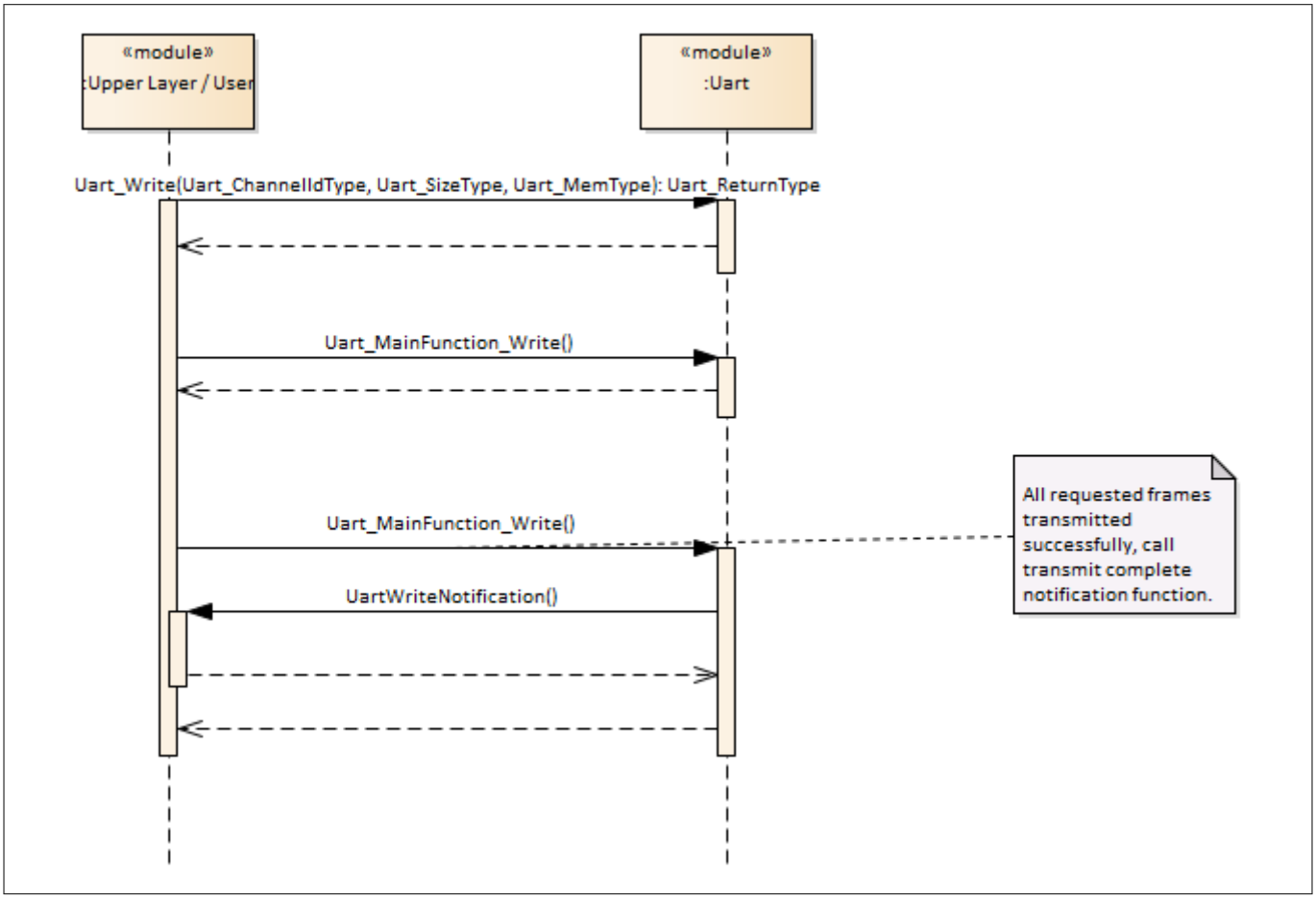


Figure 12 UART transmit operation in polling mode

The sample configuration for transmitting 8-bit frame in the polling mode is shown as follows:

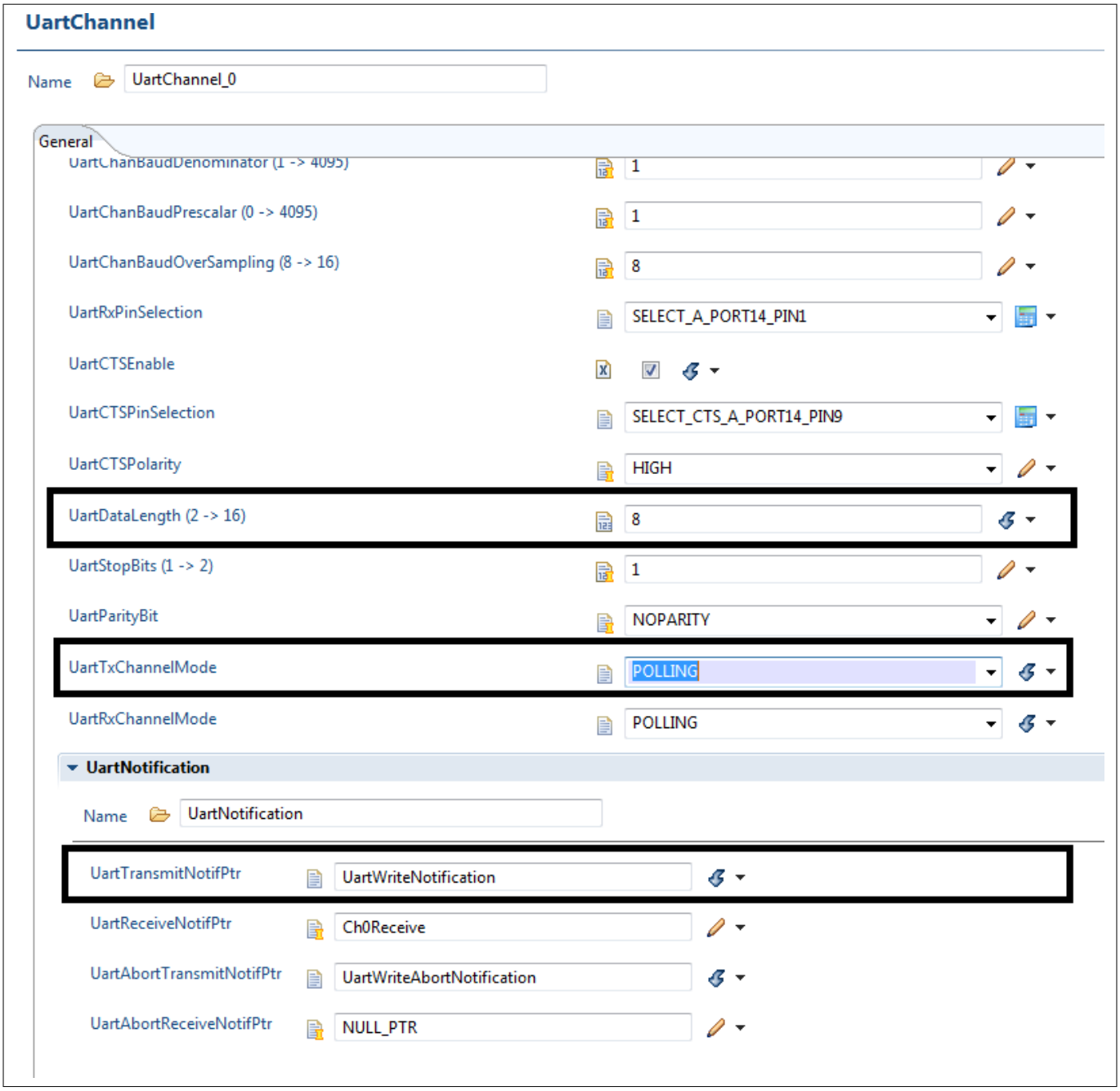


Figure 13 **Configuration: Frame length 8 bits, transmit in polling mode, transmit Notification function `UartWriteNotification`**

A sample code for transmitting 20 frames with 8-bit frame size in the polling mode is as follows:

```
/* Buffer use for transmission */
uint8 TxBuffer[20] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};

/* Write notification function */
void UartWriteNotification(Uart_ErrorIdType ErrorId)
{
 if(ErrorId == UART_NO_ERR)
 {
 /* Write operation completed without error */
 }
}

/* Uart write */
Uart_Write(0,&TxBuffer[0],20);
/* Poll till transmission is completed */
while(RetVal == UART_BUSY_TRANSMIT)
{
 /* Function to poll data transmission and give notification once transmition is finished */
 Uart_MainFunction_Write();
 /* Get channel 0 status */
 RetVal = Uart_GetStatus(0);
}
```

UART transmit abort operation

The sequence diagram of UART transmit abort operation is shown as follows:

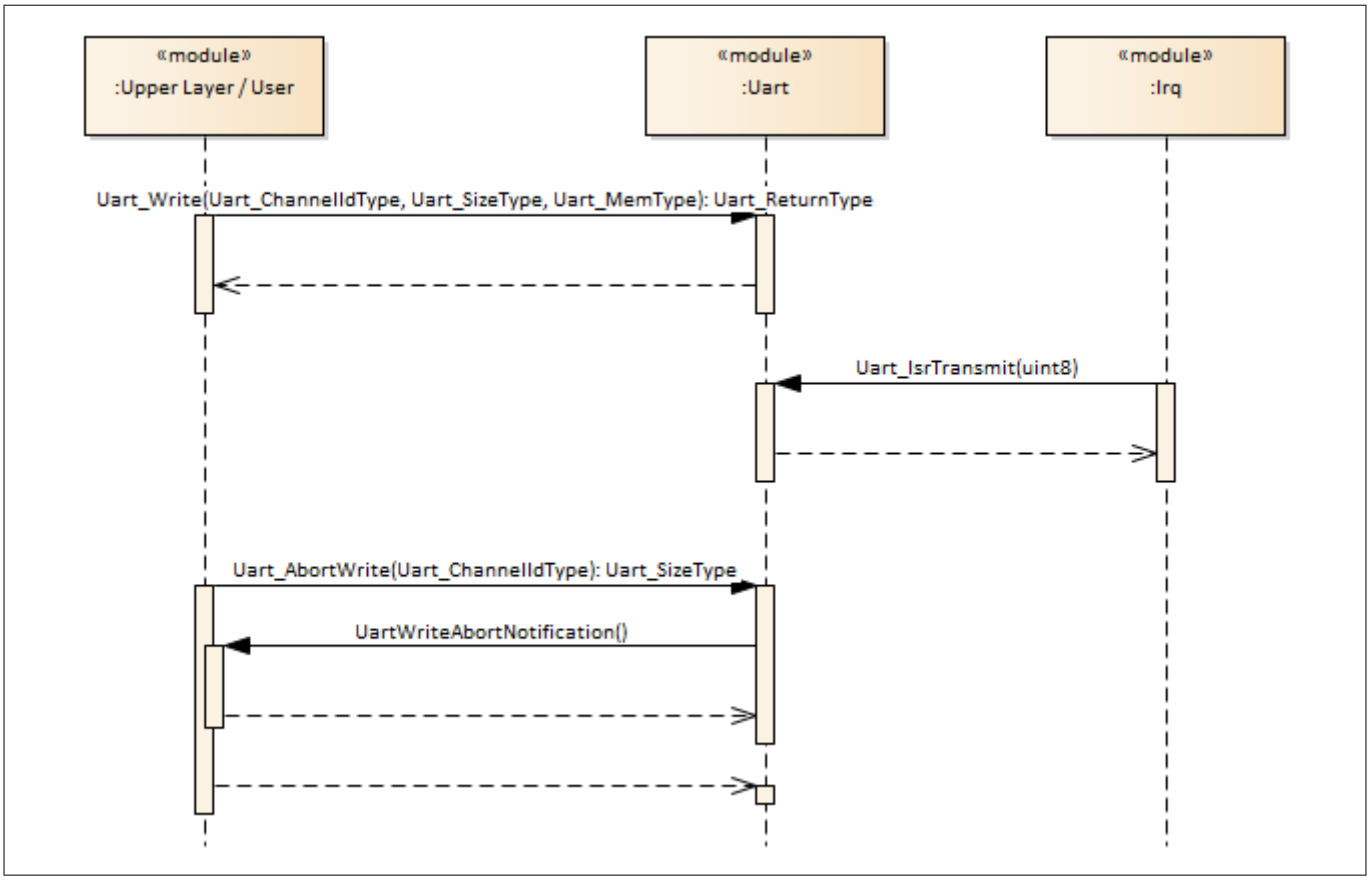


Figure 14 Transmit abort operation

A sample code for the abort transmit operation is as follows:

```
/* Buffer use for transmission */
uint8 TxBuffer[128];
uint16 NumberOfBytesTransmited;

/* Write notification function */
void UartWriteAbortNotification(Uart_ErrorIdType ErrorId)
{
 if(ErrorId == UART_NO_ERR)
 {
 /* transmit operation aborted successfully */
 }
}

/* Initialize TxBuffer */
for(Counter = 0; Counter < 128; Counter++)
{
 TxBuffer[Counter] = Counter;
}

/* Uart write */
Uart_Write(0,&TxBuffer[0],128);

/* Abort write operation on channel 0 */
NumberOfBytesTransmited = Uart_AbortWrite(0);
```

UART receive operation in interrupt mode

The sequence diagram for the UART receive operation in the interrupt mode is shown as follows:

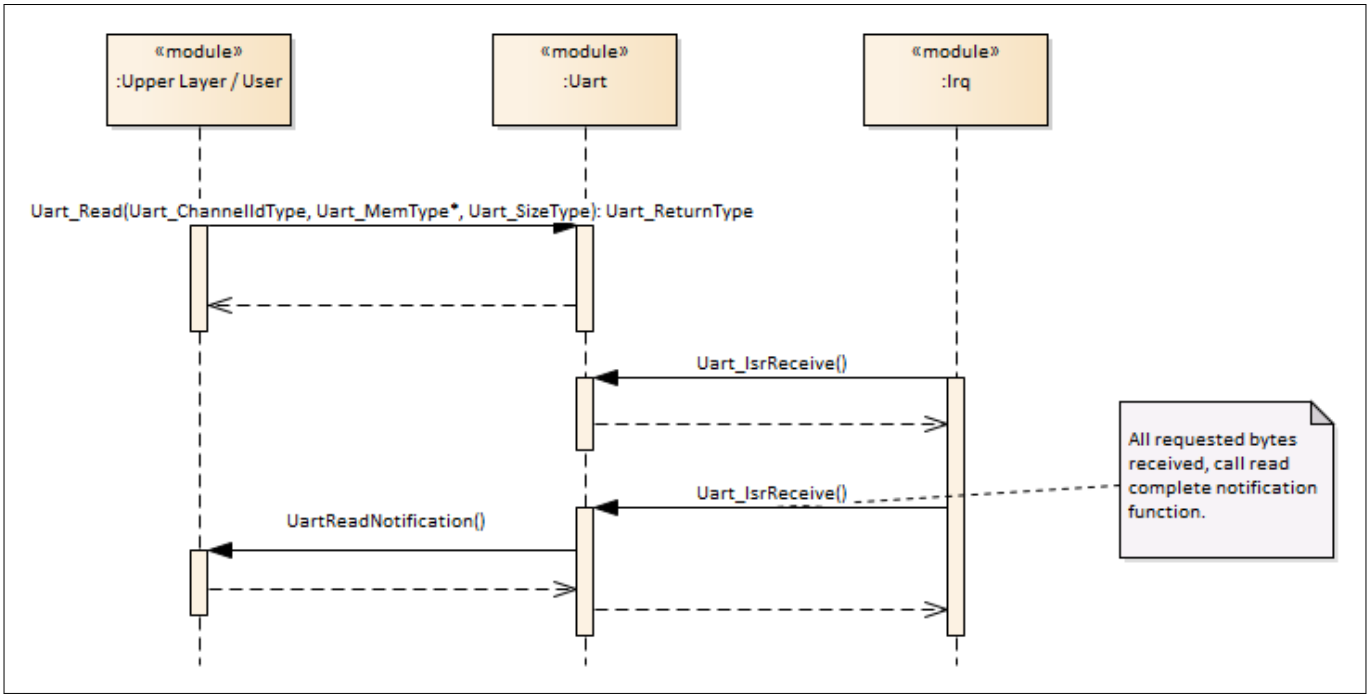


Figure 15 UART receive operation in interrupt mode

The sample configuration for receive 8-bit frame in the interrupt mode is as follows:

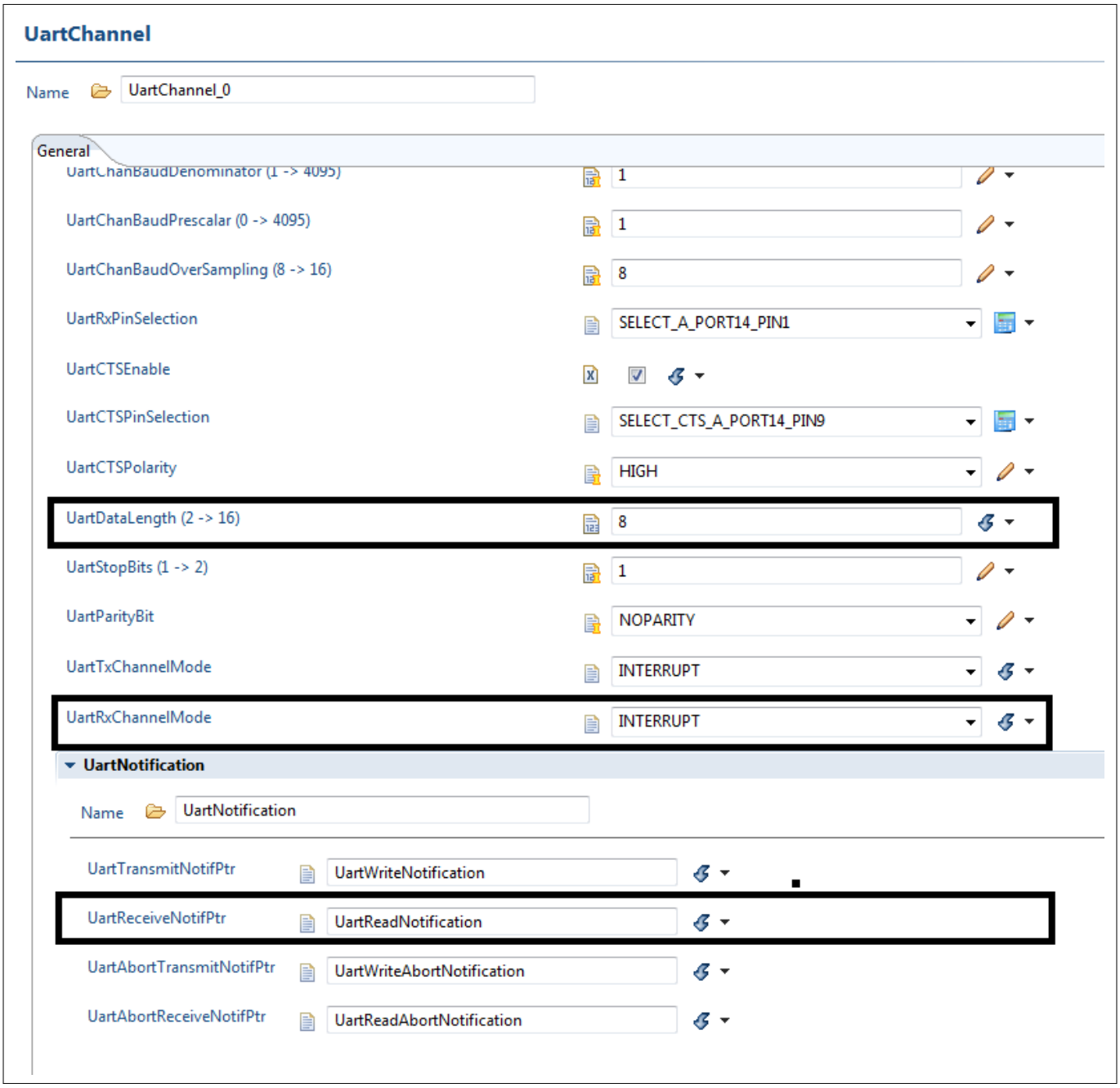


Figure 16 Configuration: Frame length 8 bits, receive in interrupt mode, receive notification function `UartReadNotification`

A sample code for receiving 20 frames with 8-bit frame size in the interrupt mode is as follows:

```
/* Receive buffer */
uint8 RxBuffer[20];

/* Read notification function */
void UartReadNotification(Uart_ErrorIdType ErrorId)
{
 if(ErrorId == UART_NO_ERR)
 {
 /* 20 frames received without error start process received data */
 }
}


/* Uart read */
Uart_Read(0,&RxBuffer[0],20);
```

UART receive operation in polling mode

The sequence diagram for the UART receive operation in the polling mode is shown as follows:

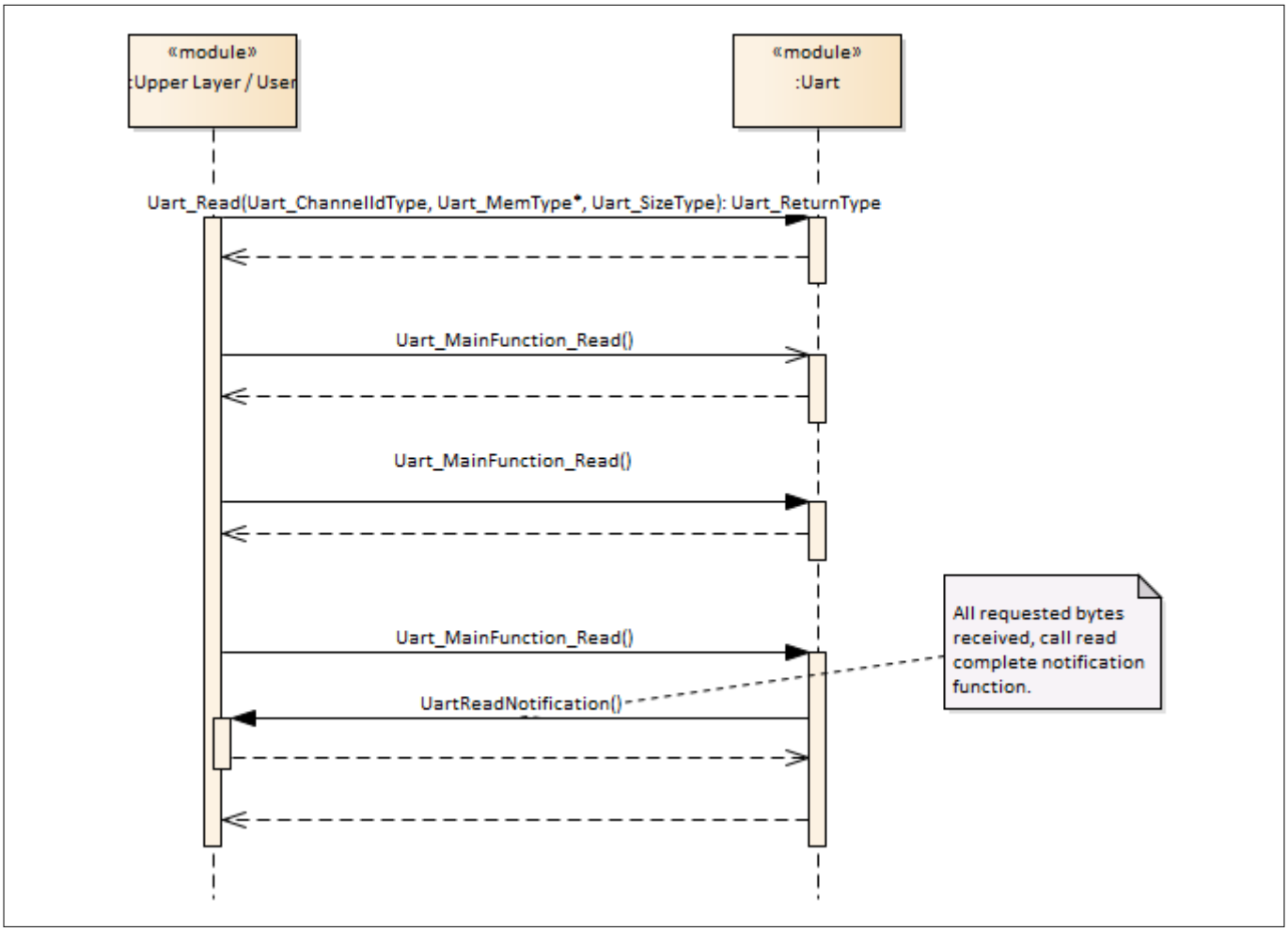


Figure 17 UART receive operation in polling mode

A sample code for receiving 20 frames with 8-bit frame size in the polling mode is as follows:

```
/* Receive buffer */
uint8 RxBuffer[20];

/* Read notification function */
void UartReadNotification(Uart_ErrorIdType ErrorId)
{
 if(ErrorId == UART_NO_ERR)
 {
 /* 20 frames received without error start process received data */
 }
}


/* Uart read */
Uart_Read(0,&RxBuffer[0],20);
/* Poll till transmission is completed */
while(RetVal == UART_BUSY_RECEIVE)
{
 /* Function to poll data receive and give notification once receivce operation is finished */
 Uart_MainFunction_Read();
 /* Get channel 0 status */
 RetVal = Uart_GetStatus(0);
}
```

UART abort read operation

The sequence diagram for the UART receive abort operation in the interrupt mode is shown as follows:

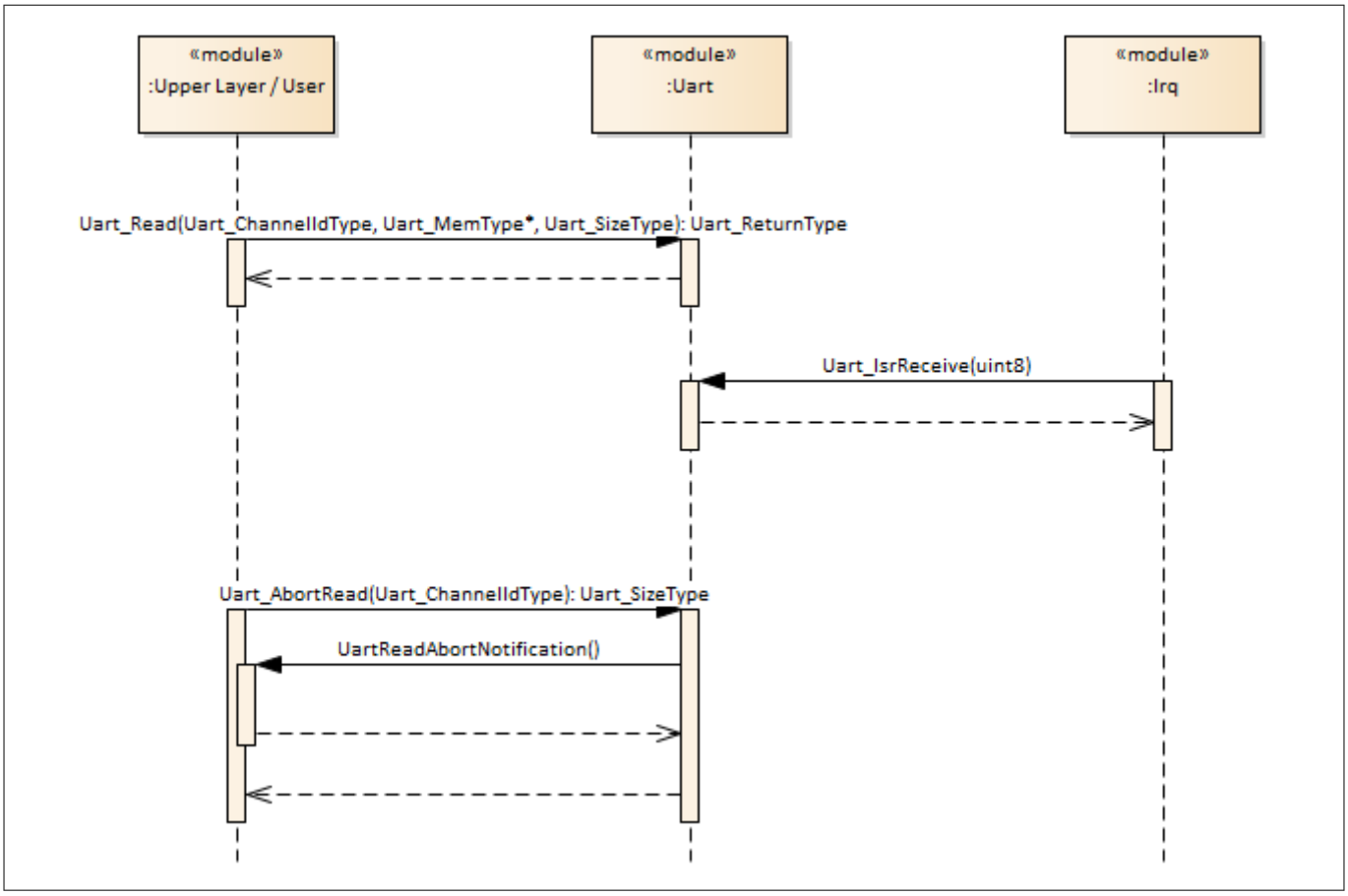


Figure 18 UART abort receive operation

A sample code for the abort receive operation is as follows:

```
/* Receive buffer */
uint8 RxBuffer[128];
uint16 NumberOfBytesReceived;

/* Read abort notification function */
void UartReadAbortNotification(Uart_ErrorIdType ErrorId)
{
 if(ErrorId == UART_NO_ERR)
 {
 /* Read operation aborted successfully */
 }
}

/* Uart read */
Uart_Read(0,&RxBuffer[0],128);

/* Abort read operation on channel 0 */
NumberOfBytesReceived = Uart_AbortRead(0);
```

UART receive streaming operation in interrupt mode

The sequence diagram for the UART receive streaming operation in the interrupt mode is shown as follows:

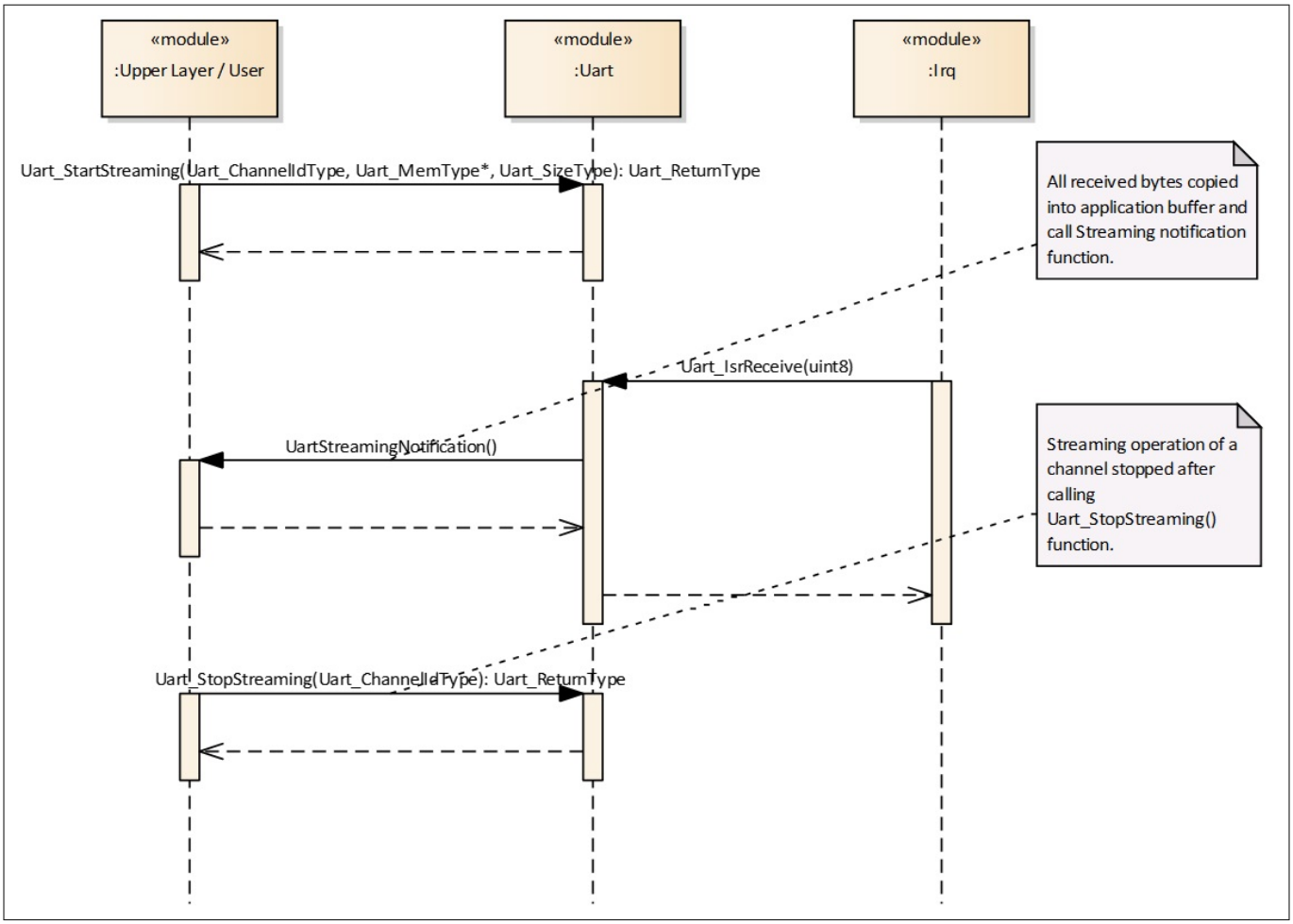


Figure 19 UART receive streaming operation in interrupt mode

The sample configuration for receive 8-bit frame in the interrupt mode is as follows:

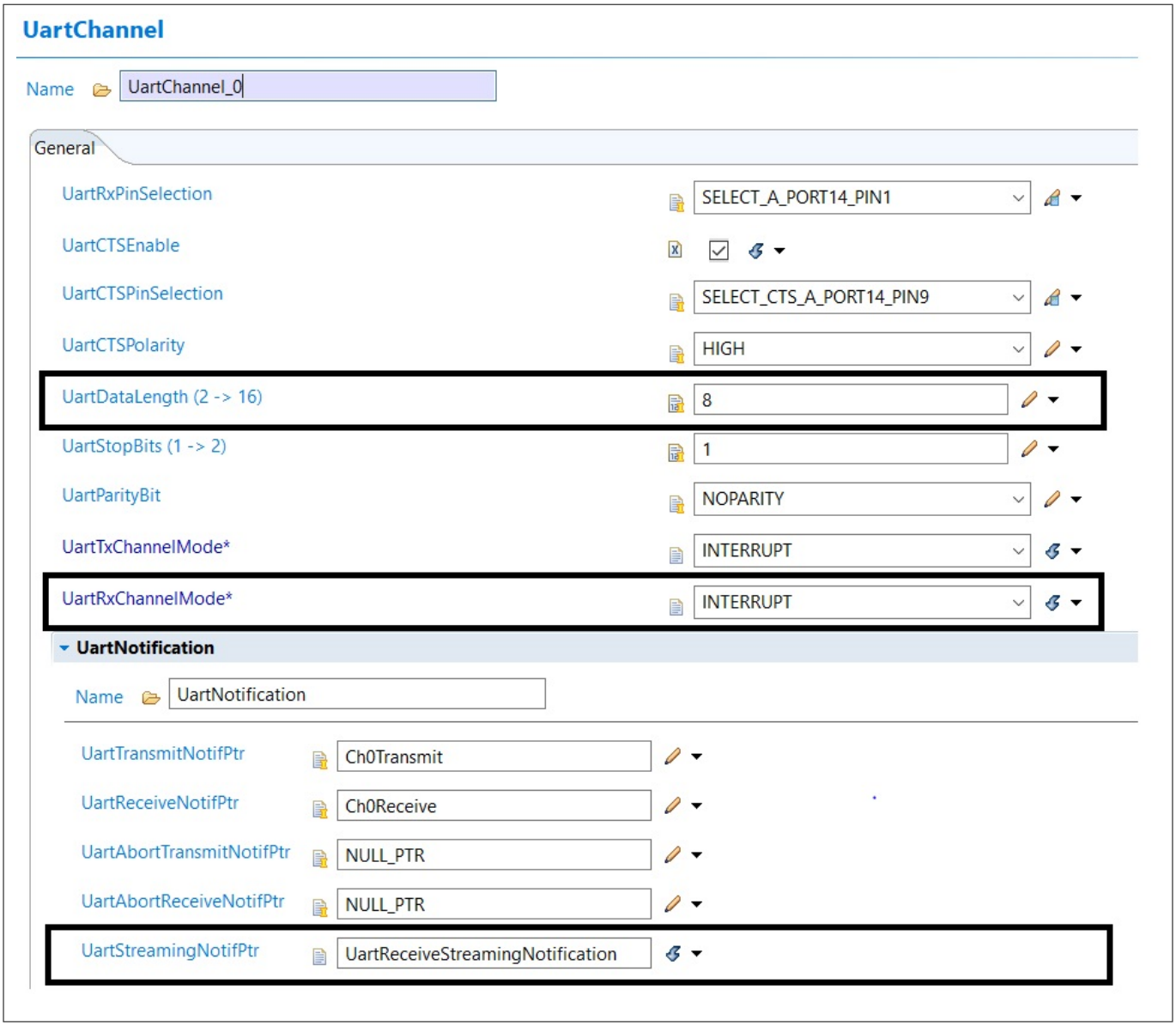


Figure 20 Configuration: Frame length 8 bits, receive streaming in interrupt mode, receive streaming notification function UartReceiveStreamingNotification

A sample code for receiving data in streaming mode with 8-bit frame size in the interrupt mode is as follows:

```
 /* Receive buffer */
 uint8 RxBuffer[16];


 /* Streaming notification function of UartChannel_0 */
 void UartReceiveStreamingNotification(Uart_ErrorIdType ErrorId, Uart_SizeType RxDataSize)
 {
 uint8 Counter;

 if(ErrorId == UART_E_NO_ERR)
 {
 /* Total received data size is given in RxDataSize parameter without error,
 start process received data */

 /* UART driver reuses the same buffer to copy next received data bytes. Hence if application
needs to process the received data later,
the application shall copy the data to another application buffer before the notification
function is returned to the UART driver */
 }
 }
 /* Uart Start streaming */
 Uart_StartStreaming(0,&RxBuffer[0],16);
```

UART receive streaming operation in polling mode

The sequence diagram for the UART receive streaming operation in the polling mode is shown as follows:

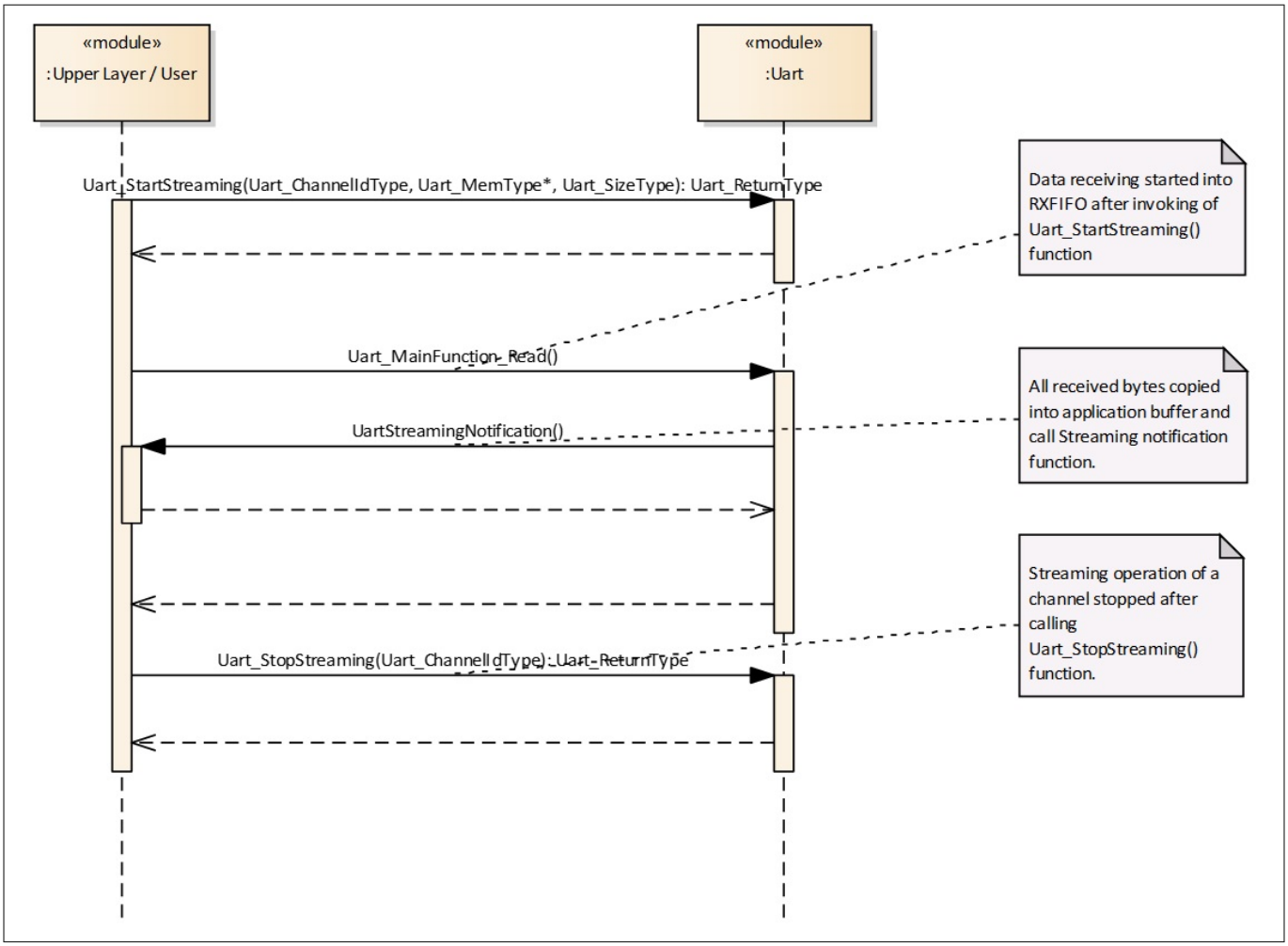


Figure 21 UART receive streaming operation in polling mode

The sample configuration for receive 8-bit frame in the polling mode is as follows:

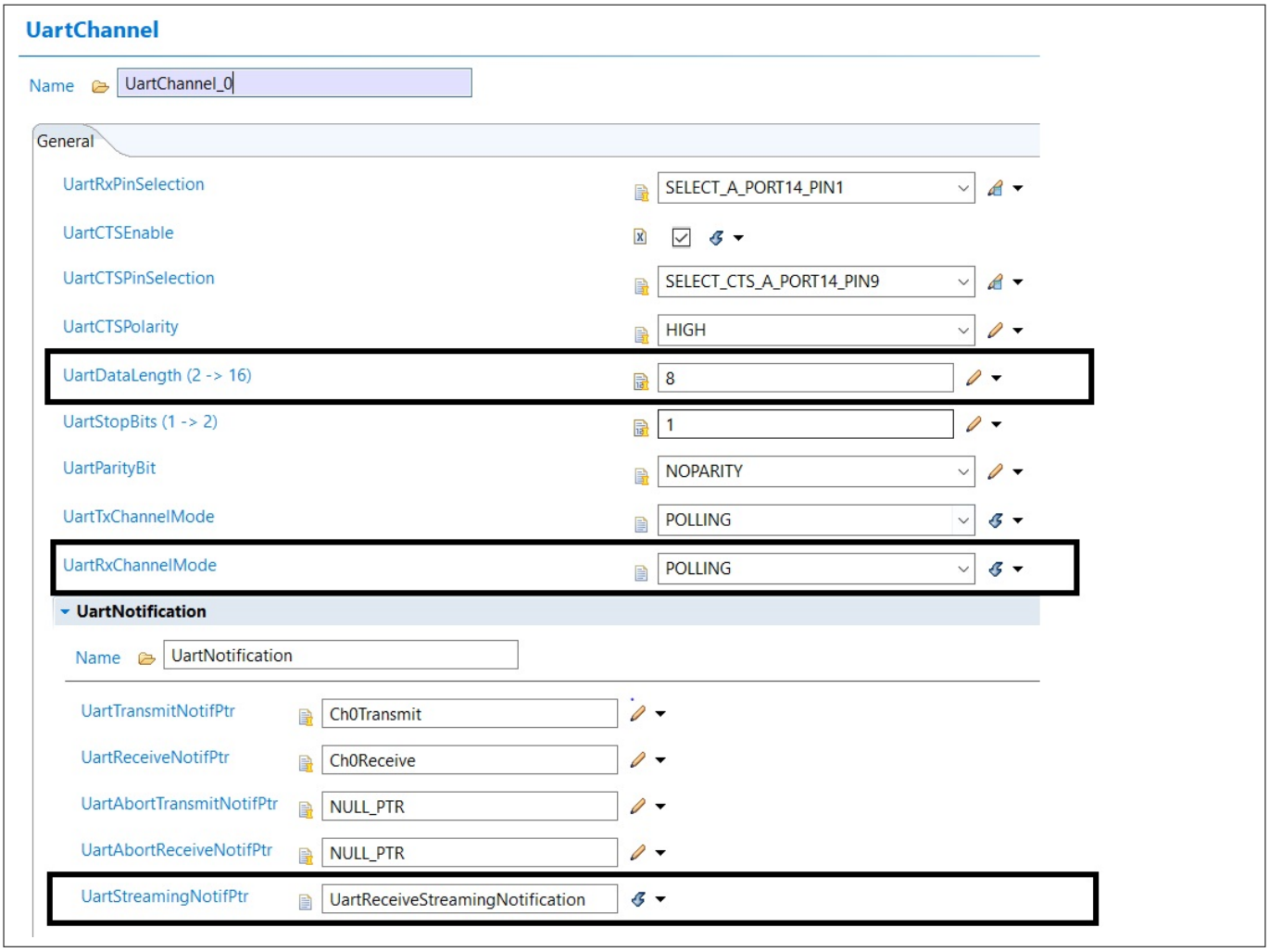


Figure 22 Configuration: Frame length 8 bits, receive in polling mode, receive streaming notification function UartReceiveStreamingNotification

A sample code for receiving data in streaming mode with 8-bit frame size in the polling mode is as follows:

```
 /* Receive buffer */
 uint8 RxBuffer[16];

 /* Streaming notification function of UartChannel_0 */
 void UartReceiveStreamingNotification(Uart_ErrorIdType ErrorId, Uart_SizeType RxDataSize)
 {
 uint8 Counter;

 if(ErrorId == UART_E_NO_ERR)
 {
 /* Total received data size is given in RxDataSize parameter without error,
 start process received data */

 /* UART driver reuses the same buffer to copy next received data bytes. Hence if application
needs to process the received data later,
 the application shall copy the data to another application buffer before the notification
function is returned to the UART driver */
 }
 }

 /* Uart Start streaming */
 Uart_StartStreaming(0,&RxBuffer[0],16);

 /* Invoke the mainfunction read with configured period */
 while(RetVal == UART_BUSY_RECEIVE)
 {
 /* Function to poll data receive and give notification once receivce operation is finished */
 Uart_MainFunction_Read();
 /* Get channel 0 status */
 RetVal = Uart_GetStatus(0);
 }
```

UART stop streaming operation

The sequence diagram for the UART stop streaming operation in the interrupt mode is shown as follows:

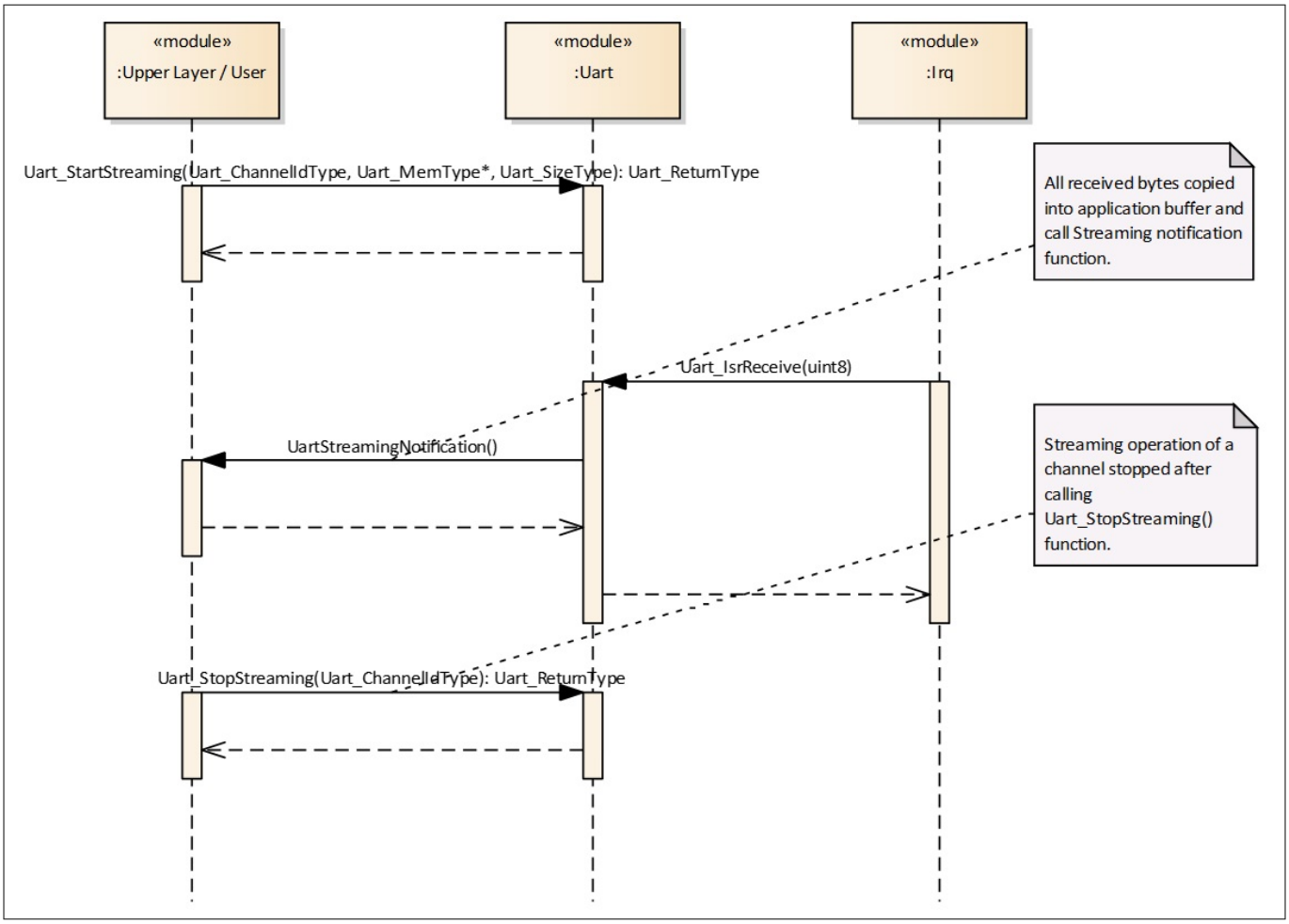


Figure 23 UART stop streaming operation

A sample code for the stop streaming operation is as follows:

```
 /* Receive buffer */
 uint8 RxBuffer[16];
 /* return value of Uart_StopStreaming API */
 Uart_ReturnType RetValue;

 /* Streaming notification function of UartChannel_0 */
 void UartReceiveStreamingNotification(Uart_ErrorIdType ErrorId, Uart_SizeType RxDataSize)
 {
 uint8 Counter;

 if(ErrorId == UART_E_NO_ERR)
 {
 /* Total received data size is given in RxDataSize parameter without error,
 start process received data */

 /* UART driver reuses the same buffer to copy next received data bytes. Hence if application
needs to process the received data later,
the application shall copy the data to another application buffer before the notification
function is returned to the UART driver */
 }
 }

 /* Uart Start streaming */
 Uart_StartStreaming(0,&RxBuffer[0],16);

 /* If expected data bytes are received or for any error reason, streaming is stopped. */
 RetValue=Uart_StopStreaming(0);
```

1.1.5 Key architectural considerations

1.2 Assumptions of Use (AoU)

The AoU for the UART driver are as follows.

- **Configuration check**

Integrator shall check that configuration code generated is correct for all the configured UART channels.
[cover parentID UART={B3D0ECC3-553D-4743-AC7A-8C6A81DEF4C0}]

- **Address check**

Integrator shall pass valid buffer pointer to transmit/receive data.
[cover parentID UART={E601C70E-216F-42eb-A2E4-DCDC329C91F1}]

- **InitCheck Sequence**

User shall invoke Uart_InitCheck to ensure the initialization is done correctly.

The parameter UartInitCheckApi shall be enabled and the user of UART shall call InitCheck function before the execution of any runtime API (except GetVersionInfo) but after completion of UART initialization sequence.
[cover parentID UART={696B916C-2ECA-480f-BA17-B60E5306EBA6}]

- **ConfigPtr passed to InitCheck**

User of UART shall ensure that InitCheck is invoked with the same ConfigPtr that is used in Init.
[cover parentID UART={870CBEF1-7406-41bf-AA0C-91C515C425DF}]

- **Freedom from interference for MCAL data**

Integrator shall ensure that there is no interference to the MCAL from other modules.

Rationale: Variables/SFRs can be corrupted by the QM software.
[cover parentID UART={ADFDA904-F0CE-431e-AC65-E1D42A402D37}]

- **Frequency check**

Baudrate parameters of ASCLIN are calculated using configured frequency. Therefore, user shall ensure that the UART driver is invoked only when the operating frequency of ASCLIN is same as the configured frequency. In case of a mismatch, the ASCLIN operates with a different baudrate than the configured value (UartBaudRate).
[cover parentID UART={4C54A9DC-9200-4126-B41F-C8E733371CEF}]

- **Receiver check**

ASCLIN cannot detect errors when data is being shifted from the shift register to the UART pins. Therefore the receiver device shall ensure to have an error detection mechanisms in place.
[cover parentID UART={85AB9E29-DE6F-49fd-B058-BE1A307BC8E5}]

- **FIFO Empty Interrupt**

Hardware triggers the TXFIFO interrupt until the last frame is shifted from TXFIFO to the shift register. Hence last frame transmission is not completed when the TXFIFO interrupt is triggered. Once the last frame transmission is completed, Transmit Complete interrupt will be triggered. However if the peripheral has to be de-initialized, the user shall wait for 1 frame duration or transmit complete notification else the last frame transmission may be stopped.
[cover parentID UART={51BE3A04-5FA3-420d-AC83-D5378ABB7003}]

- **UART driver usage mode**

It is assumed that the user of the UART driver is aware of the number of bytes to be received from the external device as the Uart_Read() API has size as a parameter. Also it is assumed that the user knows the instance of time when the data is expected to be received from external device and accordingly the Uart_Read() API is invoked. That is driver shall be operated in master configuration and not as a slave or peer device.

If the user of the UART driver is not aware of the size and instance of time of the data to be received from the external device then the user can invoke Uart_StartStreaming() API. This API requires application buffer size as a parameter from the user.

1 Uart driver

[cover parentID UART={D6E91D13-C314-435a-AAB5-716892AA30EF}]

1.3 Reference information

1.3.1 Configuration interfaces

Supported configuration variant: Post-Build

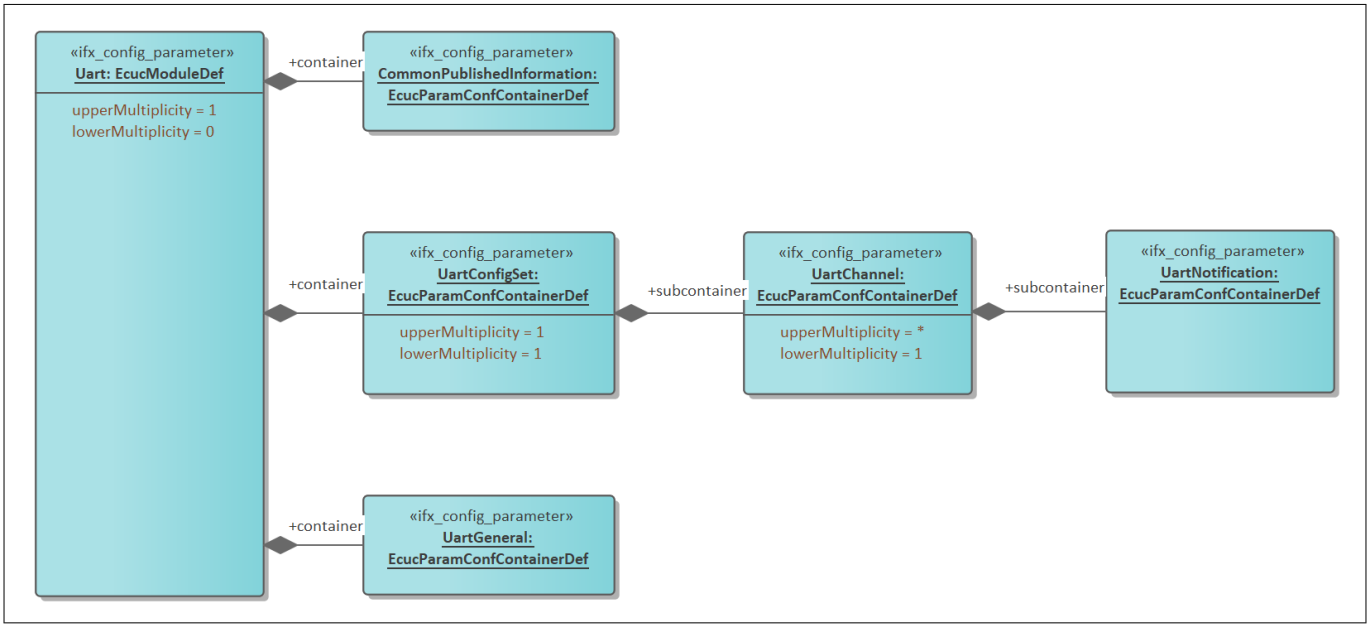


Figure 24 Container hierarchy along with their configuration parameters

1.3.1.1 Container: CommonPublishedInformation

Publish information about module.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.1.1 ArMajorVersion

Table 4 Specification for ArMajorVersion

Name	`ArMajorVersion`		
Description	Major version number of AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	4		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL

(table continues...)

Table 4	(continued) Specification for ArMajorVersion
Dependency	-
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.1.1.2 ArMinorVersion

Table 5	Specification for ArMinorVersion		
Name	`ArMinorVersion`		
Description	Minor version of AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per selected Autosar version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.3 ArPatchVersion

Table 6	Specification for ArPatchVersion		
Name	`ArPatchVersion`		
Description	Patch level version number of AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per selected Autosar version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.4 ModuleId

Table 7 Specification for ModuleId

Name	`ModuleId`		
Description	Module identifier of UART driver from module list.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	255		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.5 Release

Table 8 Specification for Release

Name	`Release`		
Description	Specifies the derivate for which the configuration project is created.		
Multiplicity	1..1	**Type**	EcucStringParamDef
Range	String		
Default value	As per UART driver.		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.6 SWMajorVersion

Table 9 Specification for SWMajorVersion

Name	`SWMajorVersion`
Description	Major version number of the implementation of the module.

(table continues...)

Table 9 (continued) Specification for SWMajorVersion

Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per driver version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.7 SWMinorVersion

Table 10 Specification for SWMinorVersion

Name	`SWMinorVersion`		
Description	Minor version number of implementation of the module.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per driver version.		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.8 SWPatchVersion

Table 11 Specification for SWPatchVersion

Name	`SWPatchVersion`		
Description	Patch level version number of implementation of the module.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per driver version		

(table continues...)

Table 11 (continued) Specification for SWPatchVersion

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.9 VendorId

Table 12 Specification for VendorId

Name	`VendorId`		
Description	Vendor identifier of dedicated implementation of UART driver according to the AUTOSAR vendor list.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	17		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2 Container: UartChannel

This container contains the configuration parameters of UART channel. Maximum number of UART channels varies as per device variant.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.2.1 UartAutoCalcBaudParams

Table 13 Specification for UartAutoCalcBaudParams

Name	`UartAutoCalcBaudParams`

(table continues...)

Table 13 (continued) Specification for UartAutoCalcBaudParams

Description	Enable or disable automatic calculation of baud rate parameters (Numerator, Denominator, Pre-scalar and Over sampling) based on the configuration of parameter UartBaudRate. User can disable the feature and manually enter the values for baud rate parameters. TRUE: Automatic calculation of baudrate parameters are enabled. FALSE: Automatic calculation of baudrate parameters are disabled.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.2 UartBaudRate

Table 14 Specification for UartBaudRate

Name	`UartBaudRate`		
Description	UART channel transmit and receive baud rate in bits per second. Parameter is applicable if UartAutoCalcBaudParams is enabled. *Note: Default value set to 9600 bits per second as UART standard baud rate.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1000 - 6250000		
Default value	9600		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.3 UartCTSEnable

Table 15 Specification for UartCTSEnable

Name	`UartCTSEnable`		
Description	Enable or disable CTS for UART channel. CTS (clear to transmit) used to notify sender that receiver is ready to receive data. TRUE: CTS is enabled. FALSE: CTS is disabled. *Note: Default CTS is disabled to save hardware resource (port pin) for basic communication.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.4 UartCTSPinSelection

Table 16 Specification for UartCTSPinSelection

Name	`UartCTSPinSelection`		
Description	This parameter selects the alternate input for the CTS select line for the given Uart channel. *Note: The first available data line for configured ASCLIN HW unit is selected as default value.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	SELECT_CTS _X_PORTY_PINZ: SELECT_CTS _X_PORTY_PINZ: This parameter varies in availability as per configured Uart channel, and device variant, where x signifies data line, Y signifies port number and Z signifies pin number. Values of X, Y, Z will be extracted from property file. For example: SELECT_CTS_A_PORT14_PIN9. SELECT_CTS _X_PORTY_PINZ: NONE: This option is chosen to indicate no CTS pin is selected for Uart driver.		
Default value	SELECT_CTS_A_PORT14_PIN9		

(table continues...)

Table 16 (continued) Specification for UartCTSPinSelection

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartCTSEnable		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.5 UartCTSPolarity

Table 17 Specification for UartCTSPolarity

Name	`UartCTSPolarity`		
Description	Parameter decides active polarity of CTS pin. Parameter applicable if UartCTSEnable is enabled. *Note: Default polarity set with HIGH.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	HIGH: CTS is considered to be active when the signal is HIGH. LOW: CTS is considered to be active when the signal is LOW.		
Default value	HIGH		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartCTSEnable		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.6 UartChanBaudDenominator

Table 18 Specification for UartChanBaudDenominator

Name	`UartChanBaudDenominator`

(table continues...)

Table 18 (continued) Specification for UartChanBaudDenominator

Description	Specifies the BRG register denominator value used for Baudrate calculation. The value configured in this parameter will be written to the BRG.DENOMINATOR register field. Baud rate is derived based on the below formula. fPD = fA / (BITCON.PRESCALER +1) fOVS = fPD * BRG.NUMERATOR / BRG.DENOMINATOR fSHIFT = fOVS / (BITCON.OVERSAMPLING + 1) fASCLINF or fASCLINS is used as input clock frequency (fA). *Note: If UartAutoCalcBaudParams is enabled then value of this parameter calculated internally. Default value set 1000 to achieve baud rates 9600 bits per second (20 MHz input frequency).*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 4095		
Default value	1000		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.7 UartChanBaudNumerator

Table 19 Specification for UartChanBaudNumerator

Name	`UartChanBaudNumerator`		
Description	Specifies the BRG register numerator value used for Baudrate calculation. The value configured in this parameter will be written to the BRG.NUMERATOR register field. Baud rate is derived based on the below formula. fPD = fA / (BITCON.PRESCALER + 1) fOVS = fPD* BRG.NUMERATOR / BRG.DENOMINATOR fSHIFT = fOVS / (BITCON.OVERSAMPLING + 1) fASCLINF or fASCLINS is used as input clock frequency (fA). *Note: If UartAutoCalcBaudParams is enabled then value of this parameter calculated internally. Default value set 24 to achieve baud rates 9600 bits per second (20 MHz input frequency).*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 4095		

(table continues...)

Table 19 (continued) Specification for UartChanBaudNumerator

Default value	24		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.8 UartChanBaudOverSampling

Table 20 Specification for UartChanBaudOverSampling

Name	`UartChanBaudOverSampling`		
Description	Specifies the BITCON register over sampling value used for Baudrate calculation. The value configured in this parameter will be written to the BITCON.OVERSAMPLING register field. Baud rate is derived based on the below formula. fPD = fA/ (BITCON.PRESCALER + 1) fOVS = fPD * BRG.NUMERATOR / BRG.DENOMINATOR fSHIFT = fOVS / (BITCON.OVERSAMPLING + 1). fASCLINF or fASCLINS is used as input clock frequency (fA). *Note: If UartAutoCalcBaudParams enabled then value of parameter calculated internally. Default value set 9 to achieve baud rates 9600 bits per second (20 MHz input frequency).*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	3 - 15		
Default value	9		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.9 UartChanBaudPrescalar

Table 21 Specification for UartChanBaudPrescalar

Name	`UartChanBaudPrescalar`		
Description	Specifies the BITCON register prescalar value used for Baudrate calculation. The value configured in this parameter will be written to the BITCON.PRESCALAR register field. Baud rate is derived based on the below formula. fPD = fA / (BITCON.PRESCALER + 1) fOVS = fPD * BRG.NUMERATOR / BRG.DENOMINATOR fSHIFT = fOVS / (BITCON.OVERSAMPLING + 1). fASCLINF or fASCLINS is used as input clock frequency (fA). *Note: If UartAutoCalcBaudParams is enabled then value of this parameter calculated internally. Default value set 4 to achieve baud rates 9600 bits per second (20 MHz input frequency).*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 4095		
Default value	4		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.10 UartChannelId

Table 22 Specification for UartChannelId

Name	`UartChannelId`		
Description	UART channel logical identifier. Upper limit of the channel identifier varies as per device variant. *Note: Minimum value of the parameter set as default value.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - *		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-

(table continues...)

Table 22 (continued) Specification for UartChannelId

Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.11 UartDataLength

Table 23 Specification for UartDataLength

Name	`UartDataLength`		
Description	Parameter decides the frame length of UART channel. *Note: Default frame size set as 8 because commonly used.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	2 - 16		
Default value	8		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.12 UartFrameSpace

Table 24 Specification for UartFrameSpace

Name	`UartFrameSpace`		
Description	Parameter decides the inter-frame space in TXFIFO. Note: The value set by this parameter indicates the number of bits to be inserted between frames. Default Inter-frame space is set as 1(indicating 1 bit frame spacing).		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 7		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-

(table continues...)

Table 24 (continued) Specification for UartFrameSpace

Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.13 UartHwUnit

Table 25 Specification for UartHwUnit

Name	`UartHwUnit`		
Description	Parameter specify ASCLIN hardware channel configured for logical channel. Maximum number of ASCLIN channel depends on device variant. *Note: Default value is set with parameter minimum value.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	ASCLIN0: Hardware channel ASCLIN0. ASCLINx: Hardware channel varies as per device variant from ASCLIN1 to ASCLINx where x is maximum number of ASCLIN channel supported by the device.		
Default value	ASCLIN0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.14 UartParityBit

Table 26 Specification for UartParityBit

Name	`UartParityBit`		
Description	Parameter decides type of parity in data frame. *Note: Default type set with no parity to reduce frame size.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef

(table continues...)

Table 26 (continued) Specification for UartParityBit			
Range	EVENPARITY: Parity bit set with 1 when even number of 1's in data frame. NOPARITY: Parity bit not present in data frame. ODDPARITY: Parity bit set with 1 when odd number of 1's present in data frame.		
Default value	NOPARITY		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.15 UartRxChannelMode

Table 27 Specification for UartRxChannelMode			
Name	`UartRxChannelMode`		
Description	UART channel receive operation configuration mode. *Note: Default set in interrupt mode to disable optional interface (schedule function will enable in case any channel configured in polling mode).*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	INTERRUPT: UART channel receive operation in interrupt mode. POLLING: UART channel receive operation in polling mode.		
Default value	INTERRUPT		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.16 UartRxPinSelection

Table 28 Specification for UartRxPinSelection	
Name	`UartRxPinSelection`

(table continues...)

Table 28 (continued) Specification for UartRxPinSelection

Description	This parameter selects the alternate input for the receive signal for the given Uart channel. *Note: The first available data line for configured ASCLIN HW unit is selected as default value.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	SELECT_X_PORTY_PINZ: SELECT_X_PORTY_PINZ: This parameter varies in availability as per configured Uart channel, and device variant, where x signifies data line, Y signifies port number and Z signifies pin number. Values of X, Y, Z will be extracted from property file. For example SELECT_A_PORT14_PIN1.		
Default value	SELECT_A_PORT14_PIN1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.17 UartStopBits

Table 29 Specification for UartStopBits

Name	`UartStopBits`		
Description	This parameter is used for selecting the number of stop bits configuration in data frame. *Note: Default value set with 1 bit to reduce frame size.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 2		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	None
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2.18 UartTxChannelMode

Table 30 Specification for UartTxChannelMode

Name	`UartTxChannelMode`		
Description	UART channel transmit operation mode. *Note: Default set in interrupt mode to disable optional interface (schedule function will enable in case any channel configured in polling mode).*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	INTERRUPT: UART channel transmit operation in interrupt mode. POLLING: UART channel transmit operation in polling mode.		
Default value	INTERRUPT		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3 Container: UartConfigSet

This container contains the channel configuration of the UART driver. This container is a multiple configuration container. This container and its sub-containers exist once per configuration set.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.4 Container: UartGeneral

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.4.1 UartAbortReadApi

Table 31 Specification for UartAbortReadApi

Name	`UartAbortReadApi`		
Description	Switch to enable or disable abort read feature. *Note: The optional APIs are disabled by default to minimize the executable code size.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef

(table continues...)

Table 31 (continued) Specification for UartAbortReadApi

Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.2 UartAbortWriteApi

Table 32 Specification for UartAbortWriteApi

Name	`UartAbortWriteApi`		
Description	Switch to enable or disable abort write feature. *Note: The optional APIs are disabled by default to minimize the executable code size.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.3 UartClockRef

Table 33 Specification for UartClockRef

Name	`UartClockRef`

(table continues...)

Table 33 (continued) Specification for UartClockRef

Description	This parameter refers to the system clock configured by MCU driver. This reference is used for BaudRate computation. *Note: Since the name of the dependent container is user configurable, the default value is kept as NULL.*		
Multiplicity	1..1	**Type**	EcucReferenceDef
Range	Reference to Node: McuAscLinChannelAllocationConf, McuClockReferencePointConfig		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.4 UartCsrClksel

Table 34 Specification for UartCsrClksel

Name	`UartCsrClksel`		
Description	This parameter selects the baud rate logic clock for the UART driver. *Note: Default value set with fast mode.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	ASCLINF: McuAscLinFastFrequency is selected as input frequency of ASCLIN. ASCLINS: McuAscLinSlowFrequencyis selected as input frequency of ASCLIN.		
Default value	ASCLINF		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.5 UartDeInitApi

Table 35 Specification for UartDeInitApi

Name	`UartDeInitApi`		
Description	Switch to enable or disable UART driver de-init feature. *Note: The optional APIs are disabled by default to minimize the executable code size.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.6 UartDevErrorDetect

Table 36 Specification for UartDevErrorDetect

Name	`UartDevErrorDetect`		
Description	Switches the Default Error Tracer (Det) detection and notification ON or OFF. TRUE: enabled (ON). FALSE: disabled (OFF).		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.7 UartIndex

Table 37 Specification for UartIndex

Name	`UartIndex`		
Description	Specifies the instance identifier of this module instance. In case single instance is present value should be 0. *Note: Default value set minimum because single instance is present.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.8 UartInitCheckApi

Table 38 Specification for UartInitCheckApi

Name	`UartInitCheckApi`		
Description	Parameter adds or removes the Uart_InitCheck() API from the code. *Note: The default value of this parameter is set to false to minimize the executable code size.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.9 UartInitDeInitApiMode

Table 39 Specification for UartInitDeInitApiMode

Name	`UartInitDeInitApiMode`		
Description	Configuration parameter defines the privilege mode in which the initialization and de-initialization API's operate. *Note: Since UART driver accesses the SFRs, it is more efficient to operate the UART driver in supervisor mode. Hence, the default mode of operation is supervisor.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	UART_MCAL_SUPERVISOR: Init and De-init APIs operate in supervisory mode. UART_MCAL_USER1: Init and De-init APIs operate in USER1 mode.		
Default value	UART_MCAL_SUPERVISOR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.10 UartMainFunctionReadPeriod

Table 40 Specification for UartMainFunctionReadPeriod

Name	`UartMainFunctionReadPeriod`		
Description	Specifies the period of main function Uart_MainFunction_Read in seconds. UART driver does not require this information but the BSW schedule will use this information.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	0 - 10.0		
Default value	0.005		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.11 UartMainFunctionWritePeriod

Table 41 Specification for UartMainFunctionWritePeriod

Name	`UartMainFunctionWritePeriod`		
Description	Specifies the period of main function Uart_MainFunction_Write in seconds. UART driver does not require this information but the BSW schedule will use this information.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	0 - 10.0		
Default value	0.005		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.12 UartRunTimeErrorDetect

Table 42 Specification for UartRunTimeErrorDetect

Name	`UartRunTimeErrorDetect`		
Description	The activation of runtime errors is configurable (ON / OFF) at pre-compile time. *Note: The detection of runtime related errors is enabled by default to ensure that runtime issues are addressed during the product lifecycle.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.13 UartSafetyEnable

Table 43 Specification for UartSafetyEnable

Name	`UartSafetyEnable`		
Description	Switch to enable or disable the safety check. TRUE: Enable safety check FALSE: Disable safety check. *Note: The detection of safety related errors is enabled by default to ensure that safety issues are addressed during the product lifecycle.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.14 UartSleepEnable

Table 44 Specification for UartSleepEnable

Name	`UartSleepEnable`		
Description	Switch enable/disable the ASCLIN module sleep request handling by setting EDIS bit in CLC register. MCU API can request for sleep mode. Refer MCU design specification for more details. TRUE: EDIS bit is set to 1 in CLC register, sleep mode request can be recognized by ASCLIN module and enter in sleep mode. FALSE: EDIS is set to 0, a sleep mode request is ignore and module continues its operation. *Note: The optional feature is disabled by default.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		

(table continues...)

Table 44 (continued) Specification for UartSleepEnable

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.15 UartStreamingRecvModeApi

Table 45 Specification for UartStreamingRecvModeApi

Name	`UartStreamingRecvModeApi`		
Description	UartStreamingRecvModeApi parameter will enable or disable Streaming mode APIs for read operation. Uart_StartStreaming and Uart_StopStreaming APIs are used for streaming operation and these are available only when this parameter value is true. *Note: The optional APIs are disabled by default to minimize the executable code size.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.16 UartTimeoutCount

Table 46 Specification for UartTimeoutCount

Name	`UartTimeoutCount`

(table continues...)

Table 46 (continued) Specification for UartTimeoutCount

Description	Specifies the maximum time in nanoseconds to wait for hardware timeout errors. *Note: UartTimeoutCount uses the STM timer current resolution and calculate maximum number of ticks to wait before expected hardware behaviour is occurred during initialization and deinitialization of Uart driver.* *Maximum value is kept as default value for this parameter.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	100 - 4294967295		
Default value	4294967295		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.17 UartVersionInfoApi

Table 47 Specification for UartVersionInfoApi

Name	`UartVersionInfoApi`		
Description	Switch to enable or disable get version information API. *Note: The optional APIs are disabled by default to minimize the executable code size.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5 Container: UartNotification

This section lists all the notification and callbacks of the Uart driver.

Post-Build Variant Multiplicity: -
Multiplicity Configuration Class: -

1.3.1.5.1 UartAbortReceiveNotifPtr

Table 48 Specification for UartAbortReceiveNotifPtr

Name	`UartAbortReceiveNotifPtr`		
Description	Parameter which holds receive abort notification function address. Definition of function present in application. If the user does not require notification then parameter shall be configured to NULL_PTR. *Note: Optional interface so default value set NULL_PTR. The UART driver does not validate the configured function name or address, User should configure valid address or function.*		
Multiplicity	1..1	**Type**	Uart_NotificationPtr Type
Range	None		
Default value	NULL_PTR		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartAbortReadApi		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5.2 UartAbortTransmitNotifPtr

Table 49 Specification for UartAbortTransmitNotifPtr

Name	`UartAbortTransmitNotifPtr`		
Description	Parameter holds transmit abort notification function address. Definition of function present in application. If the user does not require notification then parameter shall be configured to NULL_PTR. *Note: Optional interface so default value set NULL_PTR. The UART driver does not validate the configured function name or address, User should configure valid address or function.*		
Multiplicity	1..1	**Type**	Uart_NotificationPtr Type
Range	None		
Default value	NULL_PTR		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-

(table continues...)

Table 49 (continued) Specification for UartAbortTransmitNotifPtr

Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartAbortWriteApi		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5.3 UartReceiveNotifPtr

Table 50 Specification for UartReceiveNotifPtr

Name	`UartReceiveNotifPtr`		
Description	Parameter holds receive complete notification function address. Definition of function present in application. If the user does not require notification then parameter shall be configured with NULL_PTR. *Note: Optional interface so default value set with NULL_PTR. The UART driver does not validate the configured function name or address, User should configure valid address or function.*		
Multiplicity	1..1	**Type**	Uart_NotificationPtr Type
Range	None		
Default value	NULL_PTR		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5.4 UartStreamingRecvNotifPtr

Table 51 Specification for UartStreamingRecvNotifPtr

Name	`UartStreamingRecvNotifPtr`
Description	Parameter holds the streaming notification function address. Definition of function present in application. If the user does not require notification then parameter shall be configured to NULL_PTR. *Note: Default value of this configuration parameter is set to NULL_PTR. When streaming feature used for any UART receive channel then this notification should be configured with valid function name or address to provide receive notification for that channel.*

(table continues...)

Table 51 (continued) Specification for UartStreamingRecvNotifPtr

Multiplicity	1..1	**Type**	Uart_StreamingRecvNotiPtrType
Range	None		
Default value	NULL_PTR		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	UartStreamingRecvModeApi		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5.5 UartTransmitNotifPtr

Table 52 Specification for UartTransmitNotifPtr

Name	`UartTransmitNotifPtr`		
Description	Parameter holds transmit complete notification function address. Definition of function present in application. If the user does not require notification then parameter should be configured with NULL_PTR. *Note: Optional interface so default value set NULL_PTR. The UART driver does not validate the configured function name or address, User should configure valid address or function.*		
Multiplicity	1..1	**Type**	Uart_NotificationPtrType
Range	None		
Default value	NULL_PTR		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.6 Container: Uart

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: -

1.3.2 Functions - Type definitions

This section lists all the data types of the Uart driver.

1.3.2.1 Uart_ChannelIdType

Table 53 Specification for Uart_ChannelIdType

Syntax	`Uart_ChannelIdType`	
Type	uint8	
File	`Uart.h`	
Range	0-255	
Description	Data type used to specifies logical channel identifier of UART.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.2 Uart_ConfigType

Table 54 Specification for Uart_ConfigType

Syntax	`Uart_ConfigType`	
Type	Structure	
File	`Uart.h`	
Range	--	The elements of the data structure are specific to the microcontroller.
Description	Data type used to specify UART driver configuration.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.3 Uart_ErrorIdType

Table 55 Specification for Uart_ErrorIdType

Syntax	`Uart_ErrorIdType`	
Type	Enumeration	
File	`Uart.h`	
Range	0 - UART_E_NO_ERR	No error.
	1 - UART_E_PARITY_ERR	Parity error.
	2 - UART_E_FRAME_ERR	Frame error.
	3 - UART_E_RXOVERFLOW_ERR	RXFIFO overflow error.
Description	Data type specifies the error occurred during the data transmission or reception.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.4 Uart_MemType

Table 56 Specification for Uart_MemType

Syntax	`Uart_MemType`	
Type	uint8	
File	`Uart.h`	
Range	0-255	
Description	Data type of the buffer used in read and writes operation.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.5 Uart_NotificationPtrType

Table 57 Specification for Uart_NotificationPtrType

Syntax	`Uart_NotificationPtrType`
Type	Pointer to a function of type void Function_Name (const Uart_ErrorIdType ErrorId)
File	`Uart.h`
Description	Data type to specify function pointer declaration of UART call back.
Source	IFX
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.2.6 Uart_ReturnType

Table 58 Specification for Uart_ReturnType

Syntax	`Uart_ReturnType`	
Type	Enumeration	
File	`Uart.h`	
Range	0 - UART_E_OK	API successful completed.
	1 - UART_E_NOT_OK	API reported development error.
	2 - UART_E_BUSY	UART channel is busy in same operation which is requested by API.
Description	Data type used to specify the return value of Uart driver API.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.7 Uart_SizeType

Table 59 Specification for Uart_SizeType

Syntax	`Uart_SizeType`	
Type	uint16	
File	`Uart.h`	
Range	0-65535	
Description	Data type used to specify the number of bytes to be transmit or receive.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.8 Uart_StatusType

Table 60 Specification for Uart_StatusType

Syntax	`Uart_StatusType`	
Type	Enumeration	
File	`Uart.h`	
Range	0 - UART_IDLE	Idle state (no transmits or receives operation in progress).
	1 - UART_BUSY_TRANSMIT	UART channel busy in transmit operation.
	2 - UART_BUSY_RECEIVE	UART channel busy in receive operation.
	3 - UART_BUSY_TRANSMIT_RECEIVE	UART channel busy in receive and transmit operation.
Description	Data type used to specify UART channel status.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.9 Uart_StreamingRecvNotiPtrType

Table 61 Specification for Uart_StreamingRecvNotiPtrType

Syntax	`Uart_StreamingRecvNotiPtrType`
Type	Pointer to a function of type void Function_Name (const Uart_ErrorIdType ErrorId, const Uart_SizeType RxDataSize)
File	`Uart.h`
Description	Data type to specify function pointer declaration for streaming notification call back.
Source	IFX
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.2.10 UartNotificationCallback

Table 62 Specification for UartNotificationCallback

Syntax	`UartNotificationCallback`
File	`Uart.h`
Description	None
Source	IFX
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.2.11 UartStreamingRecvNotifPtr

Table 63 Specification for UartStreamingRecvNotifPtr

Syntax	`UartStreamingRecvNotifPtr`
File	`Uart.h`
Description	None
Source	IFX
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3 Functions - APIs

This section lists all the APIs of the UART driver.

1.3.3.1 Uart_InitCheck

Table 64 Specification for `Uart_InitCheck` API

Syntax	`Std_ReturnType  Uart_InitCheck`
	`(`
	`  const Uart_ConfigType * const ConfigPtr`
	`)`	

Service ID	0xD8	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Address of UART driver configuration set.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Initialization check passed. E_NOT_OK: Initialization check failed.

(table continues...)

Table 64 (continued) Specification for `Uart_InitCheck` API

Description	API returns the status of the modules initialization. API (optional API) is available only when the parameter UartInitCheckApi is enabled. *Note: Init check should be performed in the following sequence:* *1. Call Uart_Init.* *2. Call Uart_InitCheck.*
Source	IFX
Error handling	UART_E_PARAM_POINTER, UART_E_UNINIT
Configuration dependencies	UartInitCheckApi
User hints	-
SFR accessed	ASCLIN_BITCON(r), ASCLIN_BRG(r), ASCLIN_CLC(r), ASCLIN_DATCON(r), ASCLIN_FRAMECON(r), ASCLIN_IOCR(r), ASCLIN_RXFIFOCON(r), ASCLIN_TXFIFOCON(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.2 Uart_StartStreaming

Table 65 Specification for `Uart_StartStreaming` API

Syntax	`Uart_ReturnType  Uart_StartStreaming`
	`(`
	`  const Uart_ChannelIdType Channel,`
	`  Uart_MemType * const MemPtr,`
	`  const Uart_SizeType BufSize`
	`)`	

Service ID	0xE5	
Sync/Async	Asynchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different channel (Not for the same channel)	
Parameters (in)	Channel	UART channel id.
	BufSize	The length of the application buffer in bytes which is passed in the parameter MemPtr. Note: Since the hardware FIFO size is 16 bytes, at a time UART hardware can store up to 16 bytes. So recommended BufSize is 16 bytes anything above will not be utilized by the UART driver. Note: If channel frame length configured with greater than 8 bit then buffer length should be multiple of 2

(table continues...)

Table 65 (continued) Specification for `Uart_StartStreaming` API

Parameters (out)	MemPtr	Application buffer address. The UART driver uses this buffer to copy the received UART data from the hardware FIFO memory to this application buffer and invokes streaming notification function.
Parameters (in - out)	-	-
Return	Uart_ReturnType	UART_E_OK - Receive operation initiated successfully in streaming mode. UART_E_NOT_OK - Receive operation couldn't be initiated in streaming mode due to development errors. UART_E_BUSY - UART channel is busy in receive operation. If DET and Safety is disabled API will return UART_E_OK and UART_E_BUSY.
Description	API to start receiving operation in streaming mode for specified UART channel. After successful completion of this API, the UART driver will wait for the data received in to the FIFO for the requested channel. The UART driver copies the received UART data from the hardware FIFO memory to the application buffer passed as parameter to this API. Based on UART driver mode configured, the copy happens either from interrupt service routine or from the Uart_MainFunction_Read() API. After copying the data, the UART driver invokes the streaming notification function. After the notification function is returned, the UART driver reuses the same buffer to copy next received data bytes. Hence if application needs to process the received data later, the application shall copy the data to another application buffer before the notification function is returned to the UART driver. The UART driver will not monitor any time out for the reception of the data hence application shall handle the timeout. *Note: Buffer size is useful to check buffer overflow at runtime while copying received data into user specified memory.* *Note: API (optional API) is available only when the parameter UartStreamingRecvModeApi is enabled.*	
Source	IFX	
Error handling	UART_E_UNINIT, UART_E_INVALID_CHANNEL, UART_E_PARAM_POINTER, UART_E_INVALID_SIZE, UART_E_STATE_BUSY	
Configuration dependencies	UartStreamingRecvModeApi	
User hints	-	
SFR accessed	ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_RXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.3 Uart_StopStreaming

Table 66 Specification for `Uart_StopStreaming` API

Syntax	`Uart_ReturnType Uart_StopStreaming ( const Uart_ChannelIdType Channel )`	
Service ID	0xE6	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different channel (Not for the same channel)	
Parameters (in)	Channel	UART channel id.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Uart_ReturnType	UART_E_OK - Streaming operation of the channel is stopped successfully. UART_E_NOT_OK - Streaming operation of the channel couldn't stopped due to development errors or channel receive state is not UART_RX_IN_PROGRESS.
Description	API to stop streaming operation on given channel. *Note: API (optional API) is available only when the parameter UartStreamingRecvModeApi is enabled.*	
Source	IFX	
Error handling	UART_E_UNINIT, UART_E_INVALID_CHANNEL	
Configuration dependencies	UartStreamingRecvModeApi	
User hints	-	
SFR accessed	ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_RXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.4 Uart_Init

Table 67 Specification for `Uart_Init` API

Syntax	`void Uart_Init ( const Uart_ConfigType * const ConfigPtr )`	
Service ID	0xD7	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Address of UART driver configuration set.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	API to initialize all configured ASCLIN hardware units with the values referenced by the parameter ConfigPtr.	
Source	IFX	
Error handling	UART_E_ALREADY_INITIALIZED, UART_E_INIT_FAILED	
Configuration dependencies	-	
User hints	-	
SFR accessed	ASCLIN_BITCON(w), ASCLIN_BRG(w), ASCLIN_CLC(rw), ASCLIN_CSR(rw), ASCLIN_DATCON(w), ASCLIN_FRAMECON(w), ASCLIN_IOCR(w), ASCLIN_KRST0(rw), ASCLIN_KRST1(rw), ASCLIN_KRSTCLR(rw), ASCLIN_RXFIFOCON(w), ASCLIN_TXFIFOCON(w), STM_TIM0(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.5 Uart_Read

Table 68 Specification for `Uart_Read` API

Syntax	`Uart_ReturnType Uart_Read ( const Uart_ChannelIdType Channel, Uart_MemType * const MemPtr, const Uart_SizeType Size )`

(table continues...)

Table 68 (continued) Specification for `Uart_Read` API

Service ID	0xD9	
Sync/Async	Asynchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different channel (Not for the same channel)	
Parameters (in)	Channel Size	UART channel id. Number of bytes to be read. Note: If channel frame length configured with greater than 8 bit then number of bytes should be multiple of 2.
Parameters (out)	MemPtr	Application buffer address.
Parameters (in - out)	-	-
Return	Uart_ReturnType	UART_E_OK - Receive operation initiated successfully. UART_E_NOT_OK - Receive operation couldn't be initiated due to development errors. UART_E_BUSY - UART channel is busy in receive operation. If DET and Safety is disabled API will return UART_E_OK and UART_E_BUSY.
Description	API to read data from an UART channel, with specified size and the memory location. After successful completion of this API, the UART driver will wait for the data received in to the FIFO for the requested channel. The UART driver will not monitor any time out for the reception of the data hence application shall handle the timeout.	
Source	IFX	
Error handling	UART_E_UNINIT, UART_E_INVALID_SIZE, UART_E_STATE_BUSY, UART_E_INVALID_CHANNEL, UART_E_PARAM_POINTER	
Configuration dependencies	-	
User hints	-	
SFR accessed	ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_RXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.6 Uart_Write

Table 69 Specification for `Uart_Write` API

Syntax	`Uart_ReturnType Uart_Write ( const Uart_ChannelIdType Channel, const Uart_MemType * const MemPtr, const Uart_SizeType Size )`	
Service ID	0xDA	
Sync/Async	Asynchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different channel (Not for the same channel)	
Parameters (in)	Channel MemPtr Size	UART channel id. Application memory address from where data to be transmit. Number of data bytes to be transmitted. Note: If channel frame length configured with greater than 8 bits then number of bytes should be multiple of 2.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Uart_ReturnType	UART_E_OK - Transmit operation initiated successfully. UART_E_NOT_OK - Transmit operation couldn't be initiated due to development errors. UART_E_BUSY - UART channel is busy in transmit operation. If DET and Safety is disabled API will return UART_E_OK and UART_E_BUSY.
Description	API to write data to a Uart channel, with specified size and the memory location. API returning success indicates that data accepted for transmission, API will update the data to be transmitted in FIFO and enable interrupts for successive writes to FIFO.	
Source	IFX	
Error handling	UART_E_STATE_BUSY, UART_E_INVALID_SIZE, UART_E_UNINIT, UART_E_INVALID_CHANNEL, UART_E_PARAM_POINTER, UART_E_TXFIFO_FILL_ERR	
Configuration dependencies	-	
User hints	-	
SFR accessed	ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_TXDATA(w), ASCLIN_TXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	

(table continues...)

Table 69 (continued) Specification for `Uart_Write` API	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.7 Uart_AbortRead

Table 70 Specification for `Uart_AbortRead` API		
Syntax	`Uart_SizeType Uart_AbortRead ( const Uart_ChannelIdType Channel )`	
Service ID	0xDC	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different channel (Not for the same channel)	
Parameters (in)	Channel	UART channel id.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Uart_SizeType	Number of bytes successfully received and stored to the application memory location before the read operation was aborted.
Description	API to abort read operation on given channel. *Note: API (optional API) is available only when the parameter UartAbortReadApi is enabled. Abort read notification will be called at the end of successful abort.*	
Source	IFX	
Error handling	UART_E_UNINIT, UART_E_INVALID_CHANNEL	
Configuration dependencies	UartAbortReadApi	
User hints	-	
SFR accessed	ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_RXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.8 Uart_AbortWrite

Table 71 Specification for `Uart_AbortWrite` API

Syntax	`Uart_SizeType Uart_AbortWrite ( const Uart_ChannelIdType Channel )`	
Service ID	0xDB	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different channel (Not for the same channel)	
Parameters (in)	Channel	UART channel id.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Uart_SizeType	Number of bytes that have been successfully transmitted before the write operation was aborted.
Description	API to abort data transmission on given channel. *Note: API (optional API) is available only when the parameter UartAbortWriteApi is enabled Notification will be called at the end of successful abort.*	
Source	IFX	
Error handling	UART_E_UNINIT, UART_E_INVALID_CHANNEL	
Configuration dependencies	UartAbortWriteApi	
User hints	-	
SFR accessed	ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_TXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.9 Uart_GetStatus

Table 72 Specification for `Uart_GetStatus` API

Syntax	`Uart_StatusType Uart_GetStatus ( const Uart_ChannelIdType Channel )`
Service ID	0xDD

(table continues...)

Table 72 (continued) Specification for `Uart_GetStatus` API

Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different channel (Not for the same channel)	
Parameters (in)	Channel	UART channel id.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Uart_StatusType	UART_IDLE: Idle state (no transmit or receive operation in progress). UART_BUSY_TRANSMIT: UART channel busy in transmit operation. UART_BUSY_RECEIVE: UART channel busy in receive operation. UART_BUSY_TRANSMIT_RECEIVE: UART channel busy in transmit and receive operation.
Description	API to read an UART channels status.	
Source	IFX	
Error handling	UART_E_UNINIT, UART_E_INVALID_CHANNEL	
Configuration dependencies	-	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.10 Uart_DeInit

Table 73 Specification for `Uart_DeInit` API

Syntax	`void Uart_DeInit`
	`(`
	`  void`
	`)`	

Service ID	0xDE	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-

(table continues...)

Table 73 (continued) Specification for `Uart_DeInit` API

Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	UART driver de-initialization function. *Note: API (optional API) is available only if parameter UartDeInitApi is enabled.* *Upper layer need to ensure that all configured channels are in IDLE sate and no communication on the channel before driver de-initializing Uart driver.*	
Source	IFX	
Error handling	UART_E_UNINIT	
Configuration dependencies	UartDeInitApi	
User hints	-	
SFR accessed	ASCLIN_CLC(rw), ASCLIN_CSR(rw), ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_FRAMECON(w), ASCLIN_KRST0(rw), ASCLIN_KRST1(rw), ASCLIN_KRSTCLR(rw), ASCLIN_RXFIFOCON(w), ASCLIN_TXFIFOCON(w), STM_TIM0(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.11 Uart_GetVersionInfo

Table 74 Specification for `Uart_GetVersionInfo` API

Syntax	`void Uart_GetVersionInfo ( Std_VersionInfoType * const VersionInfoPtr )`	
Service ID	0xDF	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	VersionInfoPtr	Address on which version information to be stored.
Parameters (in - out)	-	-

(table continues...)

Table 74 (continued) Specification for `Uart_GetVersionInfo` API

Return	void	-
Description	API to get the version information of UART driver. *Note: API (optional API) is available only if parameter UartVersionInfoApi is enabled.*	
Source	IFX	
Error handling	UART_E_PARAM_POINTER	
Configuration dependencies	UartVersionInfoApi	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.4 Notifications and Callbacks

The UART driver does not provide any notification or callbacks.

1.3.5 Scheduled functions

This section lists all the scheduled functions of the UART driver.

1.3.5.1 Uart_MainFunction_Read

Table 75 Specification for `Uart_MainFunction_Read` API

Syntax	`void Uart_MainFunction_Read ( void )`	
Service ID	0xE0	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	Schedule function to handle receives operation in polling mode. *Note: Function will be available if any of channels receive operation configured in polling mode.*	

(table continues...)

Table 75 (continued) Specification for `Uart_MainFunction_Read` API

Source	IFX
Error handling	UART_E_FRAME_ERR, UART_E_RXFIFO_OVERFLOW, UART_E_PARITY_ERR, UART_E_INSUFFICIENT_BUFSIZE
Configuration dependencies	UartRxChannelMode
User hints	-
SFR accessed	ASCLIN_DATCON(r), ASCLIN_FLAGS(r), ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_RXDATA(r), ASCLIN_RXFIFOCON(rw) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.5.2 Uart_MainFunction_Write

Table 76 Specification for `Uart_MainFunction_Write` API

Syntax	`void Uart_MainFunction_Write ( void )`	
Service ID	0xE1	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	Schedule function to handle transmits operation in polling mode. *Note: Function will be available if any of channels transmit operation configured in polling mode.*	
Source	IFX	
Error handling	UART_E_TXFIFO_FILL_ERR	
Configuration dependencies	UartTxChannelMode	

(table continues...)

Table 76 (continued) Specification for `Uart_MainFunction_Write` API

User hints	-
SFR accessed	ASCLIN_FLAGS(r), ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(w), ASCLIN_TXDATA(w), ASCLIN_TXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.6 Interrupt service routines

This section lists all the interrupt handlers of the UART driver.

1.3.6.1 Uart_IsrError

Table 77 Specification for `Uart_IsrError` API

Syntax	`void Uart_IsrError ( const uint8 HwUnit )`	
Service ID	0xE2	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant (Not for the same HW Unit).	
Parameters (in)	HwUnit	ASCLIN channel number.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	IsrErrorHandler is invoked when any error during UART reception is triggered or when transmit complete event is triggered. Note that these events shares the same interrupt signal line due to which events cannot be handled separately in driver. *Note: UartReceiveNotifPtr triggers when receive error occurred and UartTransmitNotifPtr triggers after successful transmission of data.*	
Source	IFX	
Error handling	UART_E_RXFIFO_OVERFLOW, UART_E_PARITY_ERR, UART_E_FRAME_ERR, UART_E_INVALID_HW_UNIT, UART_E_SPURIOUS_INTERRUPT	
Configuration dependencies	UartRxChannelMode,UartTxChannelMode	

(table continues...)

Table 77 (continued) Specification for `Uart_IsrError` API

User hints	-
SFR accessed	ASCLIN_DATCON(r), ASCLIN_FLAGS(r), ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(rw), ASCLIN_RXDATA(r), ASCLIN_RXFIFOCON(rw), ASCLIN_TXFIFOCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.6.2 Uart_IsrReceive

Table 78 Specification for `Uart_IsrReceive` API

Syntax	`void Uart_IsrReceive ( const uint8 HwUnit )`	
Service ID	0xE3	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant (Not for the same HW Unit).	
Parameters (in)	HwUnit	ASCLIN channel number.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interrupt is triggered when RXFIFO filled up to configured level. *Note: RX FIFO level configured by the Uart driver and it varies from 1 to 16 bytes.*	
Source	IFX	
Error handling	UART_E_INVALID_HW_UNIT, UART_E_SPURIOUS_INTERRUPT, UART_E_INSUFFICIENT_BUFSIZE	
Configuration dependencies	UartRxChannelMode	
User hints	-	
SFR accessed	ASCLIN_DATCON(r), ASCLIN_FLAGS(r), ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(rw), ASCLIN_RXDATA(r), ASCLIN_RXFIFOCON(rw) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	

(table continues...)

Table 78 (continued) Specification for `Uart_IsrReceive` API

Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.6.3 Uart_IsrTransmit

Table 79 Specification for `Uart_IsrTransmit` API

Syntax	`void Uart_IsrTransmit ( const uint8 HwUnit )`	
Service ID	0xE4	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant (Not for the same HW Unit).	
Parameters (in)	HwUnit	ASCLIN channel number.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	Uart driver sets TXFIFO level with No of bytes to be transmitted and it varies from 1 to 16 bytes. Transmit interrupt is generated when the TXFIFO becomes empty. *Note: UartTransmitNotifPtr will trigger after successful transmission of data.*	
Source	IFX	
Error handling	UART_E_SPURIOUS_INTERRUPT, UART_E_INVALID_HW_UNIT, UART_E_TXFIFO_FILL_ERR	
Configuration dependencies	UartTxChannelMode	
User hints	-	
SFR accessed	ASCLIN_FLAGS(r), ASCLIN_FLAGSCLEAR(w), ASCLIN_FLAGSENABLE(rw), ASCLIN_TXDATA(w), ASCLIN_TXFIFOCON(rw) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.7 Callout

The Uart driver does not provide any callout.

1.3.8 Errors Handling

This section describes the various error types reported by the UART driver.

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
UART_E_PARITY_ERR: This runtime error is reported when the parity check fail.	IFX	0x01	RUNTIME	0x01	RUNTIME
UART_E_FRAME_ERR: This runtime error is reported when the frame check fail.	IFX	0x02	RUNTIME	0x02	RUNTIME
UART_E_RXFIFO_OVERFLOW: This runtime error is reported when the RXFIFO overflow error set.	IFX	0x03	RUNTIME	0x03	RUNTIME
UART_E_INSUFFICIENT_BUFSIZE: This runtime error is reported when the buffer size is less than the received data size. Note: This is applicable only for streaming channel.	IFX	0x04	RUNTIME	0x04	RUNTIME
UART_E_UNINIT: API service used without UART driver initialization.	IFX	0x00	DET_SAFETY	0x00	DET_SAFETY
UART_E_INVALID_CHANNEL: API service used with an invalid channel identifier. Uart_AbortRead API called to stop streaming operation. Uart_StopStreaming API called to stop Uart_Read operation.	IFX	0x01	DET_SAFETY	0x01	DET_SAFETY
UART_E_PARAM_POINTER: API service used with NULL pointer.	IFX	0x02	DET_SAFETY	0x02	DET_SAFETY
UART_E_STATE_BUSY: API service called when channel is in busy state. Uart_Read API called when Uart_StartStreaming on going and vice versa.	IFX	0x03	DET_SAFETY	0x03	DET_SAFETY
UART_E_INIT_FAILED: UART driver initialization fails.	IFX	0x04	DET_SAFETY	0x04	DET_SAFETY
UART_E_INVALID_SIZE: API Service called with invalid data length parameter.	IFX	0x05	DET_SAFETY	0x05	DET_SAFETY

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
UART_E_ALREADY_INITIALIZED: UART driver is already initialized.	IFX	0x06	DET_SAFETY	0x06	DET_SAFETY
UART_E_SPURIOUS_INTERRUPT: Spurious interrupt detected.	IFX	0x07	SAFETY	0x07	SAFETY
UART_E_INVALID_HW_UNIT: IRQ handler called with invalid hardware unit identifier.	IFX	0x08	SAFETY	0x08	SAFETY
UART_E_TXFIFO_FILL_ERR: TXFIFO fill error, Fill level not matched with number of bytes filled in TXFIFO.	IFX	0x09	SAFETY	0x09	SAFETY

1.3.9 Deviations and limitations

This section describes the deviations and limitations of the UART driver.

1.3.9.1 Deviations

This section describes the deviations of the UART driver.

1.3.9.1.1 Software specification deviations

The UART driver does not have any deviations.

1.3.9.1.2 AMDC Violations

The UART driver does not have any AMDC violation.

1.3.9.1.3 VSMD Violations

The UART driver does not have any VSMD violation.

1.3.9.2 Limitations

This section describes the limitations of the UART driver.

Table 80 Known limitations

Reference	Limitation

(table continues...)

Table 80 (continued) Known limitations

Uart transmit complete notification.	It is observed that the UART channel transmit complete notification is triggered before transmission of the last frame from the ASCLIN kernel. If application has any use case of calling `Uart_DeInit` API immediately after receiving the transmit completion notification, it is being observed that the receiver is unable to receive the last frame. This behavior is observed in both interrupt and polling mode. Workaround: Provide one frame delay (based on the baudrate used) before calling the `Uart_DeInit` API. Example: In 9600 kbps baud rate configuration, if the application software is waiting for transmit complete notification to invoke `Uart_DeInit` API, there should be a delay of 1.04167 milliseconds between the application received transmit complete notification and `Uart_DeInit` API invocation.
Consecutive call to Uart_Write() just after Uart_AbortWrite().	Uart_Write() called just after Uart_AbortWrite() can lead to traps and this is because an on-going UART transmission cannot be aborted in HW. e.g. if Uart_Write() is called immediately after Uart_AbortWrite() and the last byte in the ASC shift-register is being put on the bus (in which case the TCE flag is set), then a transmit-complete notification interrupt is received on this aborted channel(because the on-going transmission cannot be aborted in HW). This can interrupt the Uart_Write() which is being executed and can lead to unexpected driver behavior (traps), as the Tx Buffer pointer is set to NULL by the error ISR when transmit complete notification interrupt is received. To avoid such boundary conditions, the recommendation to the user is to wait at least of 1 byte transmission (depending on configured baud), OR wait for UART transmit complete notification, whichever happens earliest, before calling Uart_Write() on the aborted channel.

(table continues...)

Table 80 (continued) Known limitations

`Uart_Write()` blocked due to `Uart_IsrTransmit()`.	When using `Uart_Write()` to transmit low number of bytes especially of the order of 2 bytes, in some cases it is observed that even before `Uart_Write()` execution is completed, it is interrupted by its own `Uart_IsrTransmit()` interrupt. In case if `Uart_Write()` is interrupted just after `Uart_lWrite()` and before channel state is set to `UART_TX_IN_PROGRESS`, then the `Uart_IsrTransmit` interrupt is detected as a spurious interrupt and transmissions come to a halt(if safety is enabled, then `UART_E_SPURIOUS_INTERRUPT` is reported). This scenario is a boundary condition, and of rare occurrence because the last statement of the `Uart_lWrite()` is to enable transmission(refer set bit of TXFIFOCON.ENO) and the very next statement is a return to `Uart_Write()` upon which the channel status is set to `UART_TX_IN_PROGRESS`. Since `Uart_Write()` is not implementing any critical sections by default, it is recommended that the user wraps the `Uart_Write()` with a critical section(suspend and resume interrupt APIs available from the OS) only in case the user encounters this issue in their target system.

Revision history

Table 81 Major changes since last version

Date	Version	Description
2024-08-02	6.0	Document is released.
2024-07-22	5.1	• 1.1.3.1 C file structure section Figure 2, inclusion link updated for Mcal_Wrapper.h and Det.h. • 1.2 Assumptions of Use(AoU) section, updated description and heading for "Transmit Complete Notification" to " FIFO Empty Interrupt". • 1.3.1.2.12 UartFrameSpace section added for Uart TXFIFO Inter-frame space configuration. • 1.3.9.2 Limitations section, added limitation for "Uart_Write() blocked due to Uart_IsrTransmit()".
2023-12-18	5.0	Document is released.
2023-12-08	4.1	• 1.3.9.2 Limitations section Table 79 Known limitations updated with the following change: - Added limitation: Consecutive call to Uart_Write() just after Uart_AbortWrite().
2023-06-10	4.0	Document is released.
2023-05-26	3.1	• ASIL level field changed to Safety level with description as "refer to release notes" for all APIs under 1.3.3 Functions - APIs,1.3.5 Scheduled functions and 1.3.6 Interrupt service routines. • In 1.1.4 Integration hints section, the following points are modified - DEM module section has been removed. - Mcal_wrapper module section has been added. - Updated DET section to remove runtime error from the description. • Updated Figure 1 under 1.1.2 Hardware-software mapping, DEM Module is removed and Mcal_wrapper Module is added. • Updated section 1.1.3.1 C file structure to remove Dem.h and include Mcal_wrapper.h.
2021-11-17	3.0	Document is released.

(table continues...)

Table 81 (continued) Major changes since last version

2021-11-12	2.1	• Added streaming operation information in Notifications and callbacks section • Updated description for RXFIFO level interrupt and ERR or transmit complete flags • Added example usage for receive streaming operation in interrupt mode • Added example usage for receive streaming operation in polling mode • Added example usage for stop streaming • Added Uart_StartStreaming and Uart_StopStreaming APIs • Data type Uart_StreamingRecvNotiPtrType is added • UartStreamingRecvNotifPtr added under Uart notification parameters. • UartStreamingRecvModeApi parameter added • Uart Streaming notification information added under notification handling section • UART driver usage mode AoU updated • Description of UART_E_INVALID_CHANNEL and UART_E_STATE_BUSY error is updated • Added UART_E_INSUFFICIENT_BUFSIZE runtime error. • Parity and Frame error names updated in error description table
2020-11-25	2.0	Document is released.
2020-11-19	1.1	• Port support section updated. • Default value updated for UartRunTimeErrorDetect. • Range updated for UartParitybit parameter.
2020-08-13	1.0	Document is released.
2020-08-07	0.1	• Initial version. • UART chapter moved from MCISAR_TC3xx_UM_CD to thisdocument. • Updated description for Uart_IsrError interrupt handler. • Figure 5 updated in MCU support.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2024-08-02

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2024 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-ocr1484806431059

Important notice

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.